Spring 5-2013

Automated Filtering and Attribution of Archive Bathymetry Based on A Priori Knowledge

Rodney Wade Ladner
University of Southern Mississippi

The University of Southern Mississippi

AUTOMATED FILTERING AND ATTRIBUTION OF ARCHIVE BATHYMETRY BASED ON A PRIORI KNOWLEDGE

by

Rodney Wade Ladner

A Thesis
Submitted to the Graduate School
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

Approved:

[redacted]
Director

[redacted]

[redacted]

[redacted]

[redacted]
Dean of the Graduate School

May 2013

ABSTRACT

AUTOMATED FILTERING AND ATTRIBUTION OF ARCHIVE BATHYMETRY BASED ON A PRIORI KNOWLEDGE

by Rodney Wade Ladner

May 2013

Hydrographic offices hold large volumes of historical bathymetric data. Many of these valuable datasets were collected using older generation survey systems and contain little or no metadata. Current efforts to utilize these data require human intervention to remove outliers and assess quality. This thesis develops automated algorithms based on *a priori* knowledge of existing bathymetric topography to remove errant soundings and concurrently provide an estimate of uncertainty.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF ILLUSTRATIONS

Figure

CHAPTER I

INTRODUCTION

Mapping of the oceans is accomplished by conducting hydrographic surveys. These surveys are conducted not only to collect the data needed to prepare maps for safety of navigation, but also to collect data on living marine resources, essential fish habitats, coral communities, hydrothermal vents, gas seeps, and other marine features. Presently only 10% of the Earth's oceans are mapped to modern standards (Wessel and Chandler 2011).

Bathymetry is the primary dataset measured during these surveys. Bathymetry can also be determined by other means. For example, deep ocean bathymetry can be predicted from satellite altimetry (Smith and Sandwell 1997). However, this technique is only sufficient for resolving spatial features of 25 km or larger. Because the ratio of gravity to topography varies as a function of the Earth's density in different areas of the oceans this technique is also dependent on higher resolution, regional bathymetric surveys.

1.1 Collection Methods

The primary technologies for collecting bathymetry are airborne Light Detection and Ranging (LIDAR) and Sound Navigation and Ranging (SONAR). LIDAR is the newer of the technologies. The use of LIDAR is restricted to areas with clear optical properties and depths less than 70 meters due to signal attenuation.

When SONAR is used in mapping to measure the distance from its transducer to the bottom it is known as an echo sounder. Early SONARs used in bathymetric surveys used a single vertical beam per ping and are known as Vertical Beam Echo Sounders

(VBES). SONARs which employ multiple beams per ping are known as multibeam echo sounders (MBES). This technology started in the 1950's originally for military applications. Beginning in the 1970's the US Navy further developed the technology and used it to map large swaths of the ocean floor to assist in navigation of its submarine forces. The first commercial system was fielded in 1977. It consisted of 19 beams of 2.7° each giving it a swath of 51.3° which allowed it to survey with a swath width of approximately 78% of water depth (Farr 1980). Since the early 1990's bathymetric surveying has been dominated by the use of MBES.

MBES systems transmit a broad acoustic pulse from a specially designed transducer across the full swath in an across track direction. Receive beams that are much narrower (around 1 degree depending on the system) are formed and independently establish a two way travel time of the acoustic pulse. Knowing the speed of sound through the full water column, the depth and position of the return signal can be determined from the receive angle and the two-way travel time. To accurately place each beam, a MBES requires accurate measurement of the motion of the sonar relative to a Cartesian coordinate system. The measured motion values are typically heave, pitch, roll, yaw, and heading.

The International Hydrographic Organization's (IHO) Standards for Hydrographic Surveys, Special Publication No. 44 (IHO S-44 2008) provides the minimum specifications for the collection of bathymetry. S-44 specifies both the maximum allowable Total Horizontal Uncertainty (THU) and the maximum allowable Total Vertical Uncertainty (TVU) for bathymetric measurements that are to be used for safety of navigation.

Uncertainty is used to characterize the accuracy or error in measurements (JCGM 2008). It can be thought of the following way. The difference between a measurement and the true value of the measurement is the error. Since a measurement of the true value is not completely accurate it follows that the actual error is not generally available. Instead we use the uncertainty of a measurement as the statistical likelihood of this error.

Current bathymetric survey systems provide the uncertainties with the measurements. However, prior to circa 2005, survey systems did not. Calder (2006) stated the hydrographic community has a duty to convey uncertainty in data to the end user. He also provided a methodology for assessing uncertainty on individual nodes of a bathymetric grid generated from a survey when no uncertainty was present in the source data. While he was referring to a specific bathymetric product, scientists also need this information in the survey data to facilitate decisions on suitability for use and to incorporate uncertainty into other end products.

1.2 Processing MBES Data

Many types of problems can arise in bathymetric survey data (Chandler and Wessel 2008). The problems consist of

- poor or incorrect navigation
- refraction resulting from inadequate sound velocity measurements
- errant application of sensor alignment
- inadequate motion measurement

among other problems. Automating techniques for detecting and correcting these problems in archive data is difficult. Chandler and Wessel (2008) provided a method for

detecting and removing gross navigation outliers, and obvious scaling issues. They went on to provide an assessment of VBES surveys to gridded data sets using a Reweighted Regression Analysis technique. Their work focused only on track line data for VBES surveys. Automated detection and correction of the problems encountered in MBES surveys is more difficult because of the complexity of the survey system.

Early processing of MBES data grew out of manual processing techniques of VBES data. In processing of VBES data, analysts would examine profiles of time series data along track. With the advent of MBES, the first tools allowed analysts to examine time series of swath plots and invalidate individual pings and/or beams. Tools were crude and visualization often inadequate. Many attempts to automate the process have been made but none work in all environments.

Current processing techniques, although still manual, employ a methodology that analyzes surfaces. Spatially binned or gridded datasets are generated with attribution layers allowing the analyst to employ 3-D visualization tools for analyzing the bathymetry as well as co-registered statistical surfaces such as uncertainty. These are linked to the sample data allowing users to analyze anomalous areas and invalidate outliers that errantly influence the surface. This spatial analysis facilitates product compilation.

The most common compilation method for MBES data is a Digital Bathymetric Model (DBM) (Jakobsson et al. 2012). In fact there are many global (Becker et. al. 2009; Smith and Sandwell 1997) and regional (Jakobsson et al. 2012) DBMs that are publically available. Compilations of datasets can be an iterative, labor intensive process (Becker et al. 2009). As labor cost rise, this process continues to be less and less appealing. These

compilations continue to build upon historical survey data adding the latest data with each new compilation. Thus every new project re-edits the data in preparation of use. The corrections made by analyst are rarely restored to the actual archives at the source data centers (Chandler and Wessel 2008). Hence, systematic errors continue to exist in archive data (Becker et al. 2009).

In this thesis we address two problems. First, we develop an efficient technique to remove outliers from MBES survey data using *a priori* knowledge already contained in existing DBMs. Secondly, we develop the capability to populate archived data with an assessment of uncertainty that is stored with the data for re-use.

CHAPTER II

SURVEY OF METHODS

Very little literature exists for the inverse problem of removing outliers from existing bathymetric sample data based on the existence of co-located DBMs. Most researchers start with existing sample data and clean the data using an iterative, manual editing process (Becker et al., 2009). The techniques described below apply to MBES swath data. However, there is no reason these techniques cannot be adapted for VBES surveys.

2.1 Threshold Analysis

This straight forward method compares sample survey data with a source DBM and invalidates or removes those samples which exceed a specified threshold which is expressed as a percentage of water depth. Several assumptions are made when using this technique. First the DBM is assumed to be *ground truth*. There is no assumed uncertainty in the DBM and uncertainty in the sample data, if it exists, is ignored.

Additionally, care must be taken when considering the gridded surface resolution to the approximate size of the echo sounder sample. Many publically available DBMs exist on a *very* coarse resolution such 2 Arc Minutes (Smith and Sandwell 1997) and 30 Arc Seconds (Becker et al. 2009). Modern multibeam echo sounders operate at fixed angular beam widths where the footprint resolution (f) would be approximated by:

$$f = 2 * depth * \tan(beam\ width/2) \qquad (1)$$

Thus for a 1° survey system, f in 4000 m of water would be ~70 m. When comparing samples at this frequency to DBMs at much coarser resolutions, the naturally occurring

high frequency of the underlying topography could be errantly invalidated if the threshold is set too low.

The basis for this method consists of predicting the depth, $\hat{z}$, at each sample location (x, y) in the grid. Once the predicted $\hat{z}$ is computed, it is then differenced and invalidated if the absolute value of the difference is greater than the threshold value chosen. Any $\hat{z} = f(x, y)$ interpolation method can be used. A higher order polynomial interpolation may be used to predict the $\hat{z}$ from the grid for greater accuracy (Press et al. 1992). The algorithm starts with a 16 point subgrid surrounding the sample point (i.e. m=p=4) as shown in Figure 1.

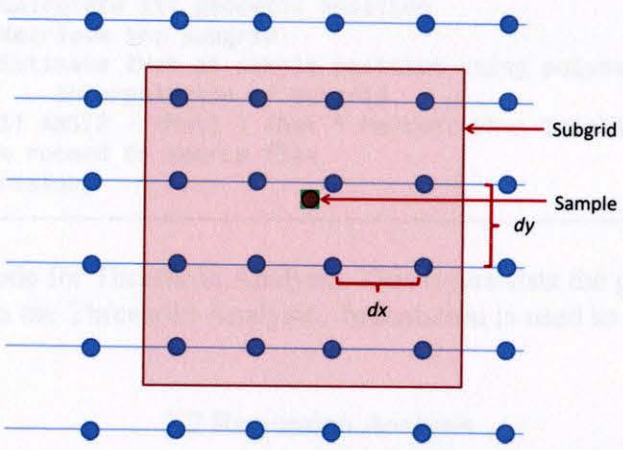

Figure 1. Sample/subgrid. This figure illustrates the relationship of the area of the subgrid, the shaded area, used in predicting the bathymetry at the sample location. Note, dx and dy are equivalent.

We implement the algorithm described in (Press et al. 1992) in C. The reference implementation uses Neville's algorithm in efficiently constructing the interpolating polynomial.

The pseudo code for this algorithm is shown in Figure 2. The interpolation is $O(mp^2)$. For n data samples where $n >> (mp^2)$ this results in an asymptotic complexity measure of $O(n)$.

Unfortunately no uncertainty estimate is determined. In fact, agreement between the samples and the DBM is inherent in the algorithm (i.e. if a sample doesn't agree, it is thrown out). Different techniques are necessary to determine not only consistency between the samples and the DBM but also inner-consistency within the samples themselves.

```
For each source file
  While not End Of File
     Read a buffer of ping records
     For each valid sample in the buffer
        Calculate its geodetic position
        Retrieve the subgrid
        Estimate ZHat at sample position using polynomial
            interpolation of subgrid
        If ABS(Z - Zhat) > Zhat * Percent then Invalidate sample
     Save record to source file
  Write history
```

Figure 2. Pseudo Code for Threshold Analysis. This figure lists the pseudo code for the algorithm to perform the Threshold Analysis. Indentation is used to indicate conditional and loop start/end.

2.2 Regression Analysis

To model the topography, one needs a modeling methodology that utilizes samples in the local area to build approximating functions that honor the local geomorphology. Such a process would allow sample comparisons with neighbors to determine inner-consistency within a sample dataset. Cleveland and Devlin (1988) specified a *Loess* local weighted regression technique that fits a surface to data in a local area. Following this technique, the framework for our topographic model is as follows. Let $(x, y)_i \ (i = 1, \ldots, n)$ be the measurement of the independent variables, longitude and

latitude respectively, and z_i $(i = 1, \ldots, n)$ be the measurements of the dependent variable depth. Then, $z_i = f(x,y)_i + \varepsilon_i$. As in most normal regressions, ε_i is assumed to be independent noise in the measurements and hence is constrained to have a mean of 0 and a variance of σ^2.

Loess regression provides an estimate $\hat{z}(x,y)$ at any (x,y) in any point in the defined domain. To define the domain, let q be an integer where $1 \leq q \leq n$. The estimate of $\hat{z}$ uses $\{(x,y)_q\}$ points closest to (x,y). Each point in the neighborhood is weighted inversely according to its' distance from (x,y). To carry out the regression, a distance function, ρ, is required. For the purposes of this work, ρ is defined as geodetic distance between a pair of points. The regression also requires a weight function and a specification of neighborhood size. The weight function used here is the tricube function given in Equation 2 where u is defined by Equation 3.

$$W(u) = \begin{cases} (1-u^3)^3, & 0 \leq u \leq 1 \\ 0, & Otherwise \end{cases} \qquad (2)$$

Let $d(x,y)$ be the distance to the q^{th} nearest $(x,y)_i$ from (x,y), then the weight of the observation at $(x,y)_i$ is:

$$u = w_i(x,y) = W\left(\frac{\rho((x,y),(x,y)_i)}{d(x,y)}\right) \qquad (3)$$

It is easy to see that $w_i(x,y)$ is a maximum closest to (x,y) and decreases to 0 as one moves away from the point to the q[th] sample. For our implementation select $q = n / 3$.

The vast majority of seafloor is gradually sloping so that a linear interpolation function in two independent variables should be adequate for processing a small set of pings over a short distance in most regions. Note, Cleveland and Devlin (1998) suggested that a quadratic fitting function be used for applications where the surface undergoes substantial curvature.

Draper and Smith (1981) used the first order linear model in two independent variables given by $z = a + bx + cy + \varepsilon$ where ε is measurement uncertainty. A fitting function, $f(x, y)$, which would estimate $\hat{z}_i = a + bx_i + cy_i + \varepsilon_i$ can be defined through the set of samples. The best fit $\mathcal{L}_2$ interpolant (the minimum least square error) with weights as defined above is given by:

$$\Pi = \sum_{i=1}^{n} w_i[z_i - f(x_i, y_i)]^2 = \sum_{i=1}^{n} w_i[z_i - (a + bx_i + cy_i)]^2 = \sum_{i=1}^{n} w_i[z_i - \hat{z}_i]^2 \quad (4)$$

The coefficients a, b and c are our unknown while the x_i, y_i, and z_i are our interpolation points. To minimize the residual, the unknown coefficients must yield a zero first derivative,

$$\begin{cases} \dfrac{\partial \Pi}{\partial a} = 2\displaystyle\sum_{i=1}^{n} w_i[z_i - (a + bx_i + cy_i)] = 0 \\ \dfrac{\partial \Pi}{\partial b} = 2\displaystyle\sum_{i=1}^{n} w_i x_i[z_i - (a + bx_i + cy_i)] = 0 \\ \dfrac{\partial \Pi}{\partial c} = 2\displaystyle\sum_{i=1}^{n} w_i y_i[z_i - (a + bx_i + cy_i)] = 0 \end{cases} \quad (5)$$

Expanding those equations nets:

$$\begin{cases} \displaystyle\sum_{i=1}^{n} w_i z_i = a\sum_{i=1}^{n} w_i 1 + b\sum_{i=1}^{n} w_i x_i + c\sum_{i=1}^{n} w_i y_i \\ \displaystyle\sum_{i=1}^{n} w_i x_i z_i = a\sum_{i=1}^{n} w_i x_i + b\sum_{i=1}^{n} w_i {x_i}^2 + c\sum_{i=1}^{n} w_i x_i y_i \\ \displaystyle\sum_{i=1}^{n} w_i y_i z_i = a\sum_{i=1}^{n} w_i y_i + b\sum_{i=1}^{n} w_i x_i y_i + c\sum_{i=1}^{n} w_i {y_i}^2 \end{cases} \quad (6)$$

Hence coefficients a, b and c can be obtained by solving the set of above linear equations.

In the current implementation, the Gnu Scientific Library functions (Free Software Foundation 2011) **gsl_multifit_wlinear** and **gsl_multifit_linear_est** are used to

solve this linear least squares system while addressing matrix condition number problems. The best-fit is found by singular value decomposition using the modified Golub-Reinsch SVD algorithm with column scaling to improve the accuracy of the singular values. Any components which have zero singular value (to machine precision) are discarded from the fit (Free Software Foundation 2011).

The algorithm is sensitive to outliers. An outlier removal step that rejects them is required (Draper and Smith 1981). If they are not rejected, the fit is heavily skewed to the outlier. The situation becomes exacerbated in the weighted locale as the increased influence on samples in the region makes detection impossible. Thus a robust technique to detect outliers prior to performing the Loess regression had to be implemented.

A Least Trimmed Squares (LTS) (Rousseeuw and Leroy 1987) approach is used due to robustness and ease of implementation. For this analysis, this objective function is constructed in the following way. Let $r_i = z_i - \hat{z}_i \; for \; i = (1, \ldots, n)$. Then

$$\Pi_{LTS} = \sum_{i=1}^{h} r_i^2 \qquad (7)$$

where $r_i{}^2$ denotes the set of ordered absolute values of the residuals (in increasing order) and $n/2 \leq h \leq n$. $h = \lceil n/2 \rceil$ was used in this implementation. In LTS, the mean of the residuals is computed from a minimum subset. Here the scaled mean of the LTS is used to filter outliers.

For this algorithm, samples need not be attributed with uncertainty at the 95% confidence level (CL). Draper and Smith (1981) provide a means for calculating the confidence limits of observations given the set of independent observations. The method is given by:

$$U = t(v, 1 - {}^{1}/_{2}\,\alpha) * s\sqrt{1 + X_0' C X_0} \qquad (8)$$

Here v are the degrees of freedom, and t is the student T distribution, s is the mean square of the residuals and $X_0' C X_0$ are the product observations and the covariance matrix. U is a type A uncertainty at the 95% CL (Taylor and Kuyatt 1994).

The second goal of this thesis was to determine how well the data set matched the reference DBM. To do this statistics are maintained on the difference of each sample and the closest node of the reference DBM. If the difference between depth at the sample location and the depth at the closest grid node is less than the computed U scaled by the distance then the sample is deemed a *good fit*.

The pseudo code to implement this technique is given in Figure 3. The overall time complexity of this method is O(n). However it does not perform nearly as quickly as the previous algorithm. The data are broken into parameterized subsets the size of which is determined by the number of pings to process through the algorithm. The number of subsamples (m) is the product of the number of pings selected by the user times the number of beams, or soundings, in a ping. While the overall process is O(n), the algorithm performs both a Least Trimmed Squares regression as well as a Loess regression in the local area using m subsamples of the overall n samples where $m << n$. Each of these processes requires a Singular Value Decomposition and matrix multiplication which are O(m^3) (Cormen et al. 2009). From examining the pseudo code it is obvious that numerous passes through m samples occur providing for O(cn) where c is a constant. Combining these would provide for an overall time complexity of O(cnm^3).

```
For each source file
  While not End Of File
    Read a buffer of ping records
    For each valid sample in the buffer
      Calculate its geodetic position
      Populate regression arrays with sample
      Perform Least Squares Fit
      For each sample Estimate Z (ZHat) and Set the residual
      Perform Least Trimmed Squares (LTS)
      For each sample Invalidate it if ZHat > scaled mean residuals from the LTS
      /* Loess Regression */
      For each valid sample
        Calculate the distance to every other valid sample
        Sort the distances
        For each sample Calculate sample weight using the Tricube function
        Perform Least Squares Fit
        Estimate ZHat, ZErr
        Calculate the Uncertainty at the 95% CL
      /* end of Loess */
      For each sample
        If invalidated then Set beamflag
        Else Calculate statistics for fit
  Write history
```

Figure 3. Regression Pseudo Code. This figure lists the pseudo code for the algorithm to perform the Regression Analysis. Indentation is used to indicate conditional and loop start/end.

CHAPTER III

CASE STUDIES

All data sets used in the case studies here are publically available. The DBM is from the United Nations Convention of the Law of the Sea project managed by the University of New Hampshire for the U. S. Department of State. The data are available at http://ccom.unh.edu/theme/law-sea. The Atlantic North Bathymetry data set is used as a reference.

We use a sample of the unprocessed survey data from the USNS Henson that includes outliers from systematic errors. By using this dataset we can compare the statistically derived 95% CL uncertainty to the total propagated uncertainty from the survey system.

We also use a R/V Knorr dataset collected by the in 1997. This data set represents data collected from a system that was two generations earlier than currently available technology, even though positioning was still done by Global Positioning System (GPS). Both multibeam data sets are available from the National Geophysical Data Center at http://www.ngdc.noaa.gov/mgg/bathymetry/multibeam.html.

Figure 4 displays the geospatial context of the data.

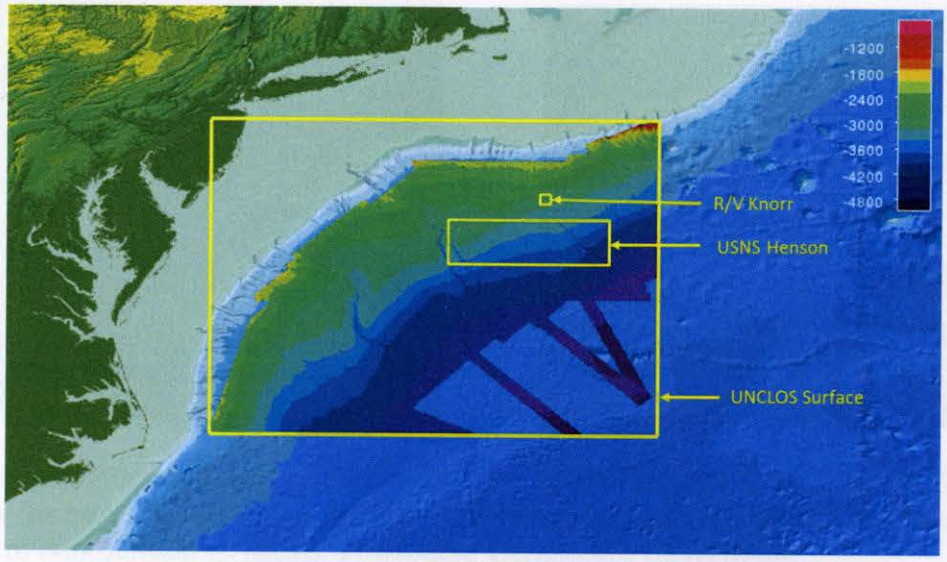

Figure 4. Data Set Coverage for Analysis. This figure illustrates the three data set used in the case studies for this thesis. The area is off the continental shelf of the US East Coast. The UNCLOS Surface is .001 arc degree spacing color coded by depth. Projection: Geographic on WGS-84.

Sample data are in Generic Sensor Format (GSF). The format is a very efficient compressed format designed specifically for MBES data. The API and format description can be found at https://www.saic.com/maritime/gsf. The Bathy/Hydro Post Processing system was used to perform geodetic calculations as needed.

3.1 Threshold Analysis

In this section we present the results from the threshold analysis. The R/V Knorr data used consisted of seven files over a period of less than two days. The initial status of the data can be seen in Figures 5a. As shown, the data are very noisy and the obvious outliers can be easily seen. The threshold analysis was run and data that deviated more than 1% from the estimated surface value from the reference DBM were invalidated. The seven data files contained 260,527 samples of which 47,282 (18.15%) were invalidated. The results are shown in Figure 5b.

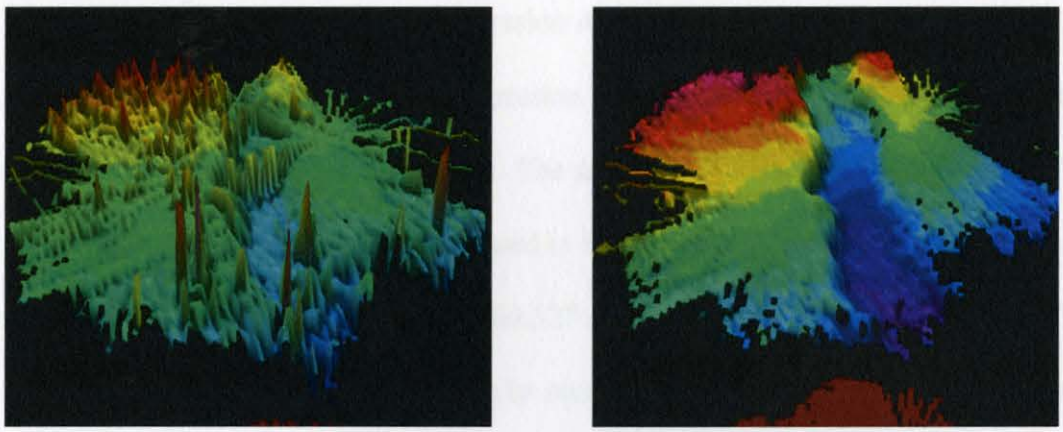

Figure 5a. Unprocessed Knorr., *Figure 5b.* Threshold processed Knorr. Figures 5a and 5b show the same dataset before and after the Threshold analysis respectively. Surfaces are 180 m bins colored by depth. Projection: Geographic onWGS-84.

Five days of USNS Henson data were processed. The initial status can be seen in Figure 6a. These data are much cleaner, however there are obvious outliers as well. From this survey, 17 files were processed using the threshold technique discussed in section 2.1 consisting of 4,318,703 samples. Of those 24,152 (0.6%) were invalidated. The results are shown in Figure 6b.

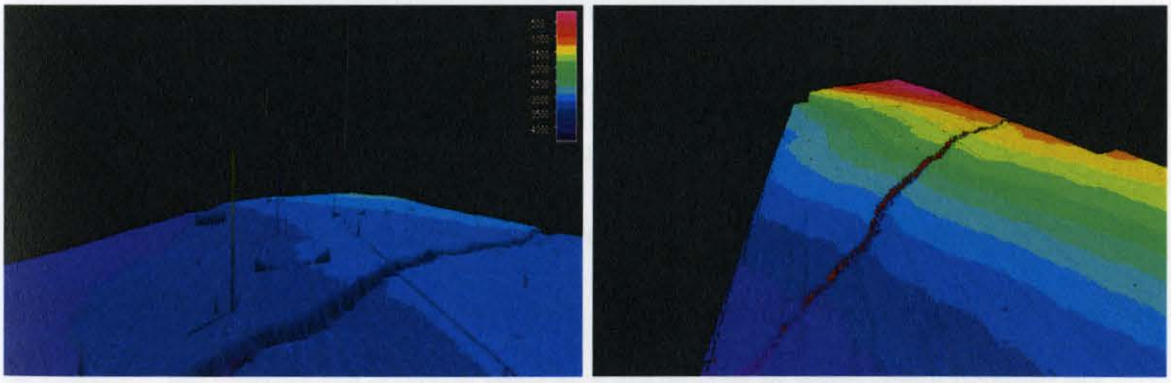

Figure 6a. Unprocessed Henson, *Figure 6b.* Threshold processed Henson. Figures 6a and 6b show the same dataset before and after the Threshold Analysis respectively. Surfaces are 180 m bins colored by depth. Projection: Geographic onWGS-84.

3.2 Regression Analysis

In this section results from our regression analysis methodology are presented and compared to the threshold analysis results. The same seven files from the R/V Knorr were processed using the technique discussed in Section 2.2. A window of ten simultaneous pings was chosen. Again, 260,527 samples were input. Of those 39,197 (15.0%) were invalidated. The results can be seen in Figure 7. Note, the shallow edited surface is shown in-lieu of the average surface (while the average surface appeared cleaner, anomalies were noted which can be viewed easier in the shallow surface.)

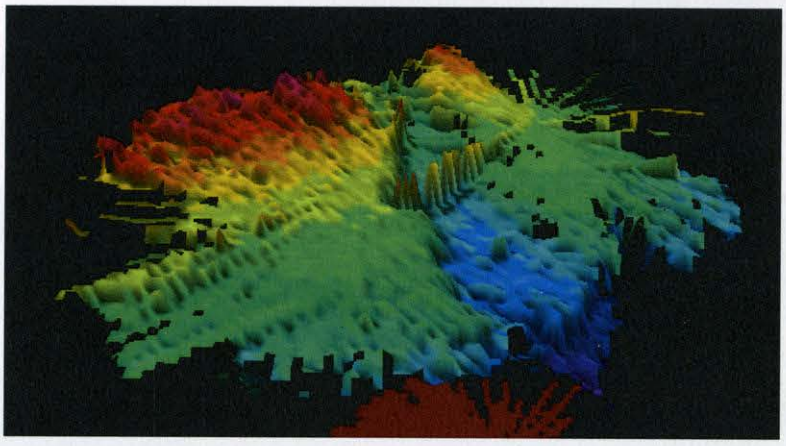

Figure 7. R/V Knorr data processed with regression. R/V Knorr data processed with the Regression Analysis technique. Shallow surface is shown. Surface is 180 m bins colored by depth. Projection: Geographic onWGS-84.

The algorithm not only filtered data but also calculated the uncertainty at the 95% CL for each sample that was not invalidated. Table 1 summarizes the results for each file processed. We express the standard error as a percentage of water depth because the error grows linearly with depth. Column 3 of Table 1 shows the average for each file. The last column contains the percentage of samples that passed the *goodness of fit* calculation with the reference DBM.

Table 1

Statistical results from R/V Knorr

File	Samples Logged	Average Standard Error (95% CL)	Percent Agreement
KNOmbN97266.d25	40,928	0.2725	81.0
KNOmbN97266.d26	35,450	0.3091	86.1
KNOmbN97266.d27	29,666	0.2674	74.8
KNOmbN97266.d28	32,628	0.2534	59.7
KNOmbN97267.d01	35,015	0.6538	88.7
KNOmbN97267.d02	31,527	0.3287	79.3
KNOmbN97267.d03	16,116	0.4589	80.5

The USNS Henson data were also processed with a ten ping window using the technique discussed in Section 2.2. Of the 4,318,703 samples read, 128,329 (3%) were invalidated. The results can be seen in Figure 8.

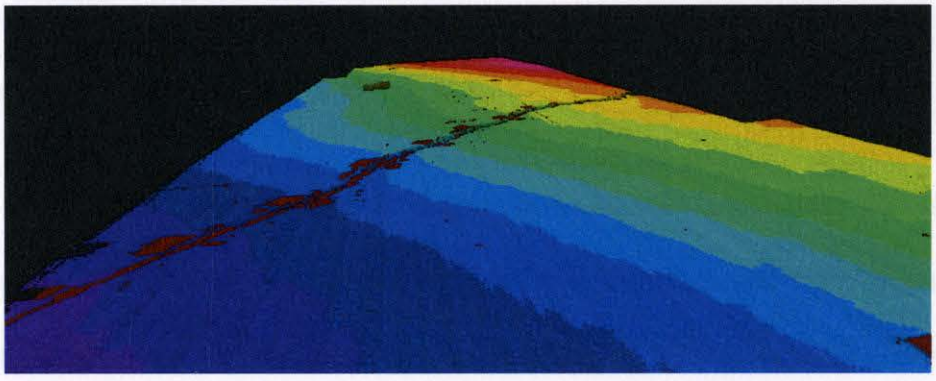

Figure 8. USNS Henson data processed with regression. USNS Henson data processed with the Regression Analysis technique. Surface is 180 m bins colored by depth. Projection: Geographic onWGS-84.

Table 2

USNS Henson Statistical Results

File	Samples Logged	U as % of depth	Percent Agreement	Uncertainty Difference (m)
63mba04278_p_100.d01	265,204	0.1286	99.0	2.25
63mba04278_p_100.d02	330,321	0.1293	98.4	2.21
63mba04278_p_100.d03	169,222	0.1348	99.1	2.07
63mba04279_p_100.d01	167,848	0.1426	98.5	1.63
63mba04279_p_100.d02	357,306	0.1425	98.7	1.78
63mba04279_p_100.d03	278,263	0.1620	98.7	1.23
63mba04280_p_100.d01	89,765	0.1586	99.4	1.08
63mba04280_p_100.d02	388,619	0.1622	98.9	1.19
63mba04280_p_100.d03	366,158	0.1609	99.0	1.30
63mba04281_p_100.d01	29,967	0.1423	97.7	1.68
63mba04281_p_100.d02	409,530	0.1565	98.5	1.46
63mba04281_p_100.d03	401,183	0.1548	98.7	1.57
63mba04281_p_100.d04	38,292	0.2192	97.4	0.91
63mba04282_p_100.d01	396,576	0.1543	98.9	1.62
63mba04282_p_100.d02	308,463	0.1597	99.1	1.52
63mba04282_p_100.d03	122,900	0.1959	98.6	0.106
63mba04282_p_100.d04	71,632	0.2164	99.0	0.602

Table 2 shows statistical results for the USNS Henson data. Column 3 has *U* expressed as a percent of water depth. Note the very high agreement in samples to the reference surface as well. The USNS Henson data has a (TVU) column for every sample calculated using the algorithm in Hare et al. (1995). The last column in Table 2 shows the mean difference between the forward propagated TVU and *U* as result of the regression. Note, the water depth ranges from ~2850 meters to ~4400 meters with the TVU in the range of 4 to 12 meters.

3.3 Run Time Analysis

CPU runtime statistics were logged for a subset of the data runs. Table 3 shows the runtimes for data files using the thresholding method. Figure 9 shows O(n) performance.

Table 3

Runtimes for Thresholding Method

File	Number of Samples (n)	Run time (secs)
63mba04278_p_100.d01	276364	3.536556
63mba04278_p_100.d02	338178	4.319485
63mba04278_p_100.d03	172191	2.199004
63mba04279_p_100.d01	177248	2.254105
63mba04279_p_100.d02	365933	4.647734
63mba04279_p_100.d03	287523	3.804542
KNOmbN97266.d25	44374	0.642294
KNOmbN97266.d26	40101	0.542826
KNOmbN97266.d27	36270	0.481337
KNOmbN97266.d28	37425	0.451381
KNOmbN97267.d01	43210	0.495069
KNOmbN97267.d02	37358	0.431122
KNOmbN97267.d03	21789	0.260553

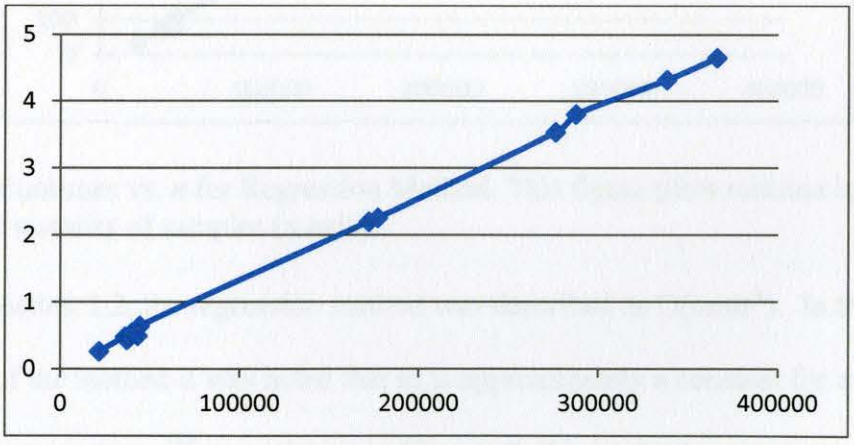

Figure 9. Runtimes vs. *n* for Thresholding Method. This figure plots runtime in seconds (y axis) vs. the quantity of samples (x axis).

Table 4

Runtimes for Regression Method

File	Number of Samples (n)	Run time (secs)
63mba04278_p_100.d01	276364	665.30734
63mba04278_p_100.d02	338178	841.518422
63mba04278_p_100.d03	172191	433.248352
63mba04279_p_100.d01	177248	413.917105
63mba04279_p_100.d02	365933	909.202196
63mba04279_p_100.d03	287523	703.180804
KNOmbN97266.d25	44374	105.965054
KNOmbN97266.d26	40101	87.363631
KNOmbN97266.d27	36270	73.072524
KNOmbN97266.d28	37425	80.446842
KNOmbN97267.d01	43210	78.530627
KNOmbN97267.d02	37358	66.859947
KNOmbN97267.d03	21789	33.591769

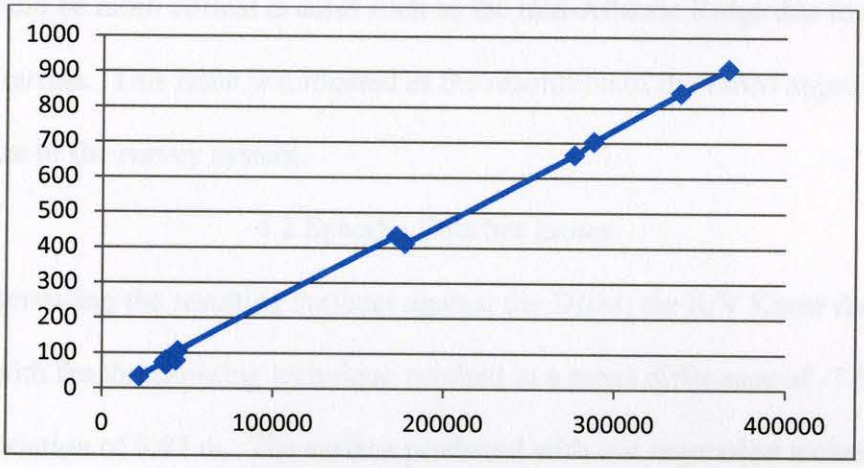

Figure 10. Runtimes vs. *n* for Regression Method. This figure plots runtime in seconds (y axis) vs. the quantity of samples (x axis).

In Section 2.2, the regression method was described as O(cnm^3). In the discussion of the method it was noted that m is approximately a constant for a particular set of input parameters. Thus as shown in Figure 10, this method scales linearly (O(n)) as well. The runtime of the algorithm includes an *albeit* large constant, m^3, that extends runtimes of this method.

CHAPTER IV

DISCUSSION

4.1 General Issues

The DBM is comprised of a grid built from a set of surveys conducted by UNH using U.S. Naval Oceanographic Office survey ships. The DBM used as reference for processing the two datasets shown in Section 3.1 was built at a grid spacing of .001° (90m) in both latitude and longitude. It should be noted that the thresholding method used is sensitive to resolution. The sensitivity arises from bathymetric aliasing that would result in the DBM as the resolution becomes more and more coarse. This sensitivity would be less critical in areas where the terrain is flat such as the reference area, but could be more critical is areas such as the mid-Atlantic Ridge due to frequent changes in terrain. This issue is mitigated as the resolution of the DBM approaches the footprint size of the survey system.

4.2 Specific Data Set Issues

Differencing the resulting surfaces against the DBM; the R/V Knorr data processed with the thresholding technique resulted in a mean difference of -7.56 m with a standard deviation of 9.97 m. The surface produced with our regression technique resulted in a mean difference of -14.26 m with a standard deviation of 24.6 m. Similar results, although much tighter, were computed by differencing the resulting USNS Henson surfaces with the reference DBM. The results showed a mean of -1.1 m with a standard deviation of 4.5 m and a mean of 1.0 m with a standard deviation of 5.4 m respectively. The threshold technique did a better job of filtering the data against the reference surface.

However, the thresholding technique did not provide any assessment of internal consistency or sample uncertainty as did the regression analysis. The filtering of the regression analysis was based on internal consistency between samples and not the underlying surface. Thus it does not have the sensitivity to the underlying resolution of the DBM, but instead is sensitive to internal errors. As is shown in Figure 11, a ping of data from the R/V Knorr survey was not invalidated. Upon further review that ping was in a turn and it's heading is approximately 180° from the preceding ping and the following ping. Thus the data, when processed as consecutive pings, is not geospatially near the bracketing pings and thus is not invalidated.

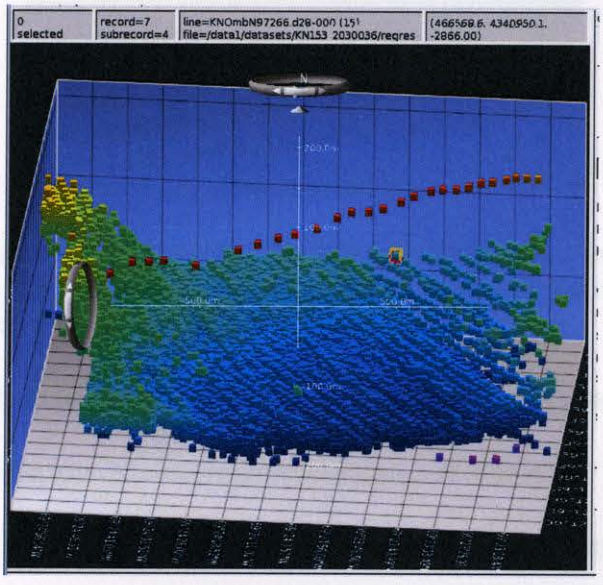

Figure 11. Errant ping in R/V Knorr. This picture illustrates a geographically misplaced ping in the R/V Knorr dataset. Samples are rendered in 3-D with geographic coordinates on the x and y axis and depth as the z axis.

A similar issue can also be found in the USNS Henson data. Figure 8 shows a very clean surface with the exception of a ridge protruding in the far left. Upon further investigation this is the result of an errant, misplaced ping. As shown in Figure 12, the errant ping, in yellow, is a lone ping from data file 63mba04282_p_100.d02. There are

no other data from that file in the locale. The light blue samples on the left are from data file 63mba04278_p_100.d02, the red is from 63mba04278_p_100.d03 and the blue on the right are from 63mba04279_p_100.d02. When processed as consecutive pings, *there was no local data to invalidate the samples* in the ping, thus it passed through the filter.

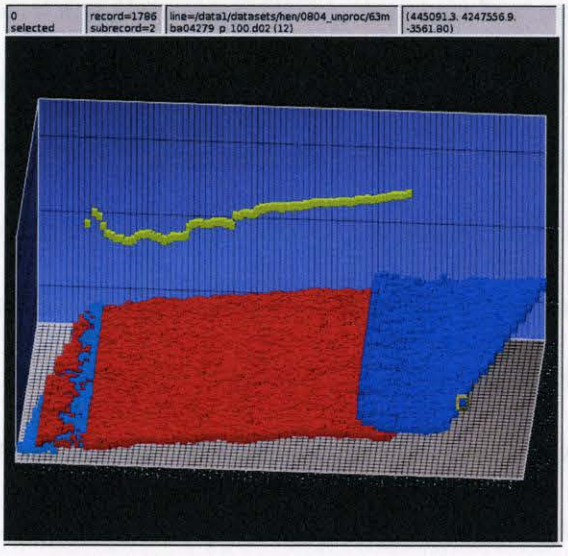

Figure 12. Errant ping in USNS Henson. This picture illustrates a geographically misplaced ping in the USNS Henson dataset. Samples are rendered in 3-D with geographic coordinates on the x and y axis and depth as the z axis.

Of particular note are the *U* calculations. These agree very well with the TVU data from the USNS Henson dataset (refer to column 3 of Table 5). The sonar manufacturer's specifications state that the vertical accuracy of the sonar is to 0.1% of water depths in sea states of 4 or less. Noting that other components of the system such as navigation, refraction, etc. contribute to the actual accuracy (Hare et al. 1995), these numbers are expected to be above 0.1% but less than 0.3%. They are all within that range.

4.3 Other Techniques

Alternative methods which predict the DBM as data are ingested could be adapted as well. Calder and Mayer (2003) specify a Kalman filter type process which estimates depth and uncertainty at grid nodes as samples are ingested. They use a Dynamic Linear Model as defined by West and Harrison (1997) to maintain best estimates of depth and variance at node points. In their method only local samples within a distance of the nodal spacing plus the horizontal error influence node estimates. Horizontal and vertical uncertainties are combined into a unified effect. The sample influence is inversely proportional to distance from the node and the magnitude of the unified uncertainty. Furthermore, samples are maintained in a fixed size queue per node sorted by depth and assimilated into the estimates from the center of the queue minimizing outlier effect and enforcing consistency. This process leads to the ability to build multiple hypotheses of depth surfaces at node locations when a sample's observed depth ± uncertainty is outside node's estimated depth ± variance.

Surfaces from this process could be used to drive the process defined in Section 2.1. The surface variance provides the filter threshold. However, applying this process to older data to define a variance in the samples would be problematic. This process assumes there exists attribution of both horizontal and vertical uncertainties on the input samples. The normal process for deriving these inputs for modern survey systems uses a forward propagated technique as defined by Hare et al. (1995). The approach could use tabulated system estimates for horizontal uncertainty (Elmore et al. 2012) and vendor provided specifications for vertical accuracy. Then, in-lieu of constructing multiple

hypothesis, samples not lying within the nodal depth ± variance would be flagged invalid and discarded.

The goals of this paper are to develop an efficient technique to remove outliers from MBES survey data and develop the capability to populate archived data with an assessment of uncertainty. The process described in this section would achieve the first goal but an estimate for sample error would remain to be derived. Additionally, some measure of agreement between the sample dataset(s) (*a priori* knowledge) would also have to be determined.

CHAPTER V

SUMMARY

5.1 Conclusions

This thesis presents the Thresholding Method for removing data outliers using the *a priori* knowledge of an existing DBM. This method is efficient and effective assuming the resolution is appropriate. However, no assessment of uncertainty is determined.

Also presented is the Regression Analysis for removing outliers, determining internal consistency and an assessment of uncertainty in the vertical. The uncertainty compares favorably with forward propagated TVU and the manufacturer's specifications for sensor performance. Little use is made of *a priori* knowledge in the filtering of outliers even though statistical measures provide for a correlation between the sample set and the DBM.

5.2 Future Work

As stated in Chapter II, the regression method scales as O(cnm^3). In this method, the Loess regression is weighted to use 1/3 of actually samples provided in the ping window. This can be scaled to use a portion of the beams within a swath, i.e. 1/2 or 1/3 of a swath. For the sample datasets, the USNS Henson has a swath consisting of 121 samples. Thus 10 pings provides a sample set of 1210 samples of which ~400 are used in the weighted Loess regression with those closest receiving much higher weights. Dividing the ping into half would have resulted in 605 samples with ~200 used in the Loess regression thus cutting m in half. This enhancement remains to be explored.

Outlier detection was performed using a Least Trimmed Squares technique as a front end to the Loess regression. As shown in Chapter III and discussed in Section 4.2,

this method is not sensitive to geospatially related data since data are processed in a time series and will not trap outliers in all situations. Furthermore, the simple thresholding technique works well as long as the spatial relationship in both resolution and coverage between the source dataset and the reference DBM holds. A combination of these outlier detection techniques would be more appropriate. The application should sense the spatial relationship up-front and if it holds then perform the thresholding for outlier detection in–lieu of Least Trimmed Squares outlier detection. However, if that relationship does not hold, then proceed with Least Trimmed Squares.

One other issue remains unaddressed at this point. In accordance with IHO S-44 (2008), all samples should be attributed with THU as well as vertical. Determining this on a sample-by-sample basis in post time is much more difficult when the exact positioning technique and mode of operation are not known. Calder (2006) suggest applying knowledge of navigation techniques used during the era of survey be used as a first estimate. Elmore et al. (2012) employed a Monte Carlo technique to establish a horizontal uncertainty estimate to train a Bayesian network. They then used the knowledge of positioning techniques for a dataset to derive a horizontal uncertainty from the associated probability density function. A similar enhancement must be added to complete the capability of this application.

APPENDIX A

MBFILTER SOURCE CODE

```
/* Module MBFilter.cpp */
/*****************************************************************************
*
* MBFilter
*
* Purpose: This program uses a priori knowledge (a grid) to invalidate outliers
*       in an observation dataset.  This is done in one of two methods
*       1. Use a higher bilinear interpolation to predict the bathymetry at
*          the sample location in the grid.  If the residual of the two is
*          outside a threshold (1% of depth is the default) then sample is
*          flagged as invalid.
*       2. It performs a Loess Regression to determine the correlation
*          between the surface and the samples.  Prior to the Loess Regression,
*          outliers are flagged are the result of a robust Least Trimmed squares
*          technique.
*
* Usage: MBFilter -(b P|r Q) gridname datafile(s)
*      where b - residuals analysis against the grid and P is the percent of
*                depth threshold
*            r - regression analysis and Q is the number of simultaneous pings to use
*                in the regression.
*
* Method:
*      1. Parse the command line arguments.
*      2. Read the grid header. Determine extents, x res, y res
*      3. Read the ping of the first MB file.


*****************************************************************************/

#include <gsf.h>
#include "MBFilter.h"

using namespace std;

/* Global accumulators */

long allBeamsRead = 0;
long allBeamsProcessed=0;
long allBeamsFiltered=0;
long allBeamsInAgreement=0;
double arg3;
int type;           /* 1 - rediual analysis, 2 - regression */
ofstream sfile;

int main ( int argc, char **argv)
{
   grid g;
   fstream file;
   string history;
   char **files=NULL;
   int nFiles=0;
   string filePath, sName;

   if (argc < 5)
   {
```

```
    usage();
    exit(-1);
}

/* get the the type of analysis */
if (strcmp (argv[1], "-b") == 0)
{
    arg3 = strtod(argv[2], NULL);
    if (arg3 <= 0.0 || arg3 > 20.0)
    {
        cout << " Error: Percentage must be a valid positive integer in the range of 1 - 5." << endl << endl;
        usage();
        exit (-1);
    }
    type = 1;
}
else if (strcmp(argv[1], "-r") == 0)
{
    arg3 = strtod(argv[2], NULL);
    if (arg3 < 5.0 || arg3 > 100.0)
    {
        cout << " Error: Pings to use must be a valid positive integer in the range of 5 - 100." << endl <<
                endl;
        usage();
        exit (-1);
    }
    type = 2;
}
else
{
    cout << " Error: you must choose resiual _(-b) or regression (-r). " << endl << endl;
    exit (-1);
}

/* biuld a list of input files */
if ((readFileList (argv[4], &files, &nFiles)) != 0)
{
    cout << "Error read input file list " << endl;
}

/* Open and read the grid file */

if (g.ingestGrid(argv[3]) != 0)
{
    /*error read ing grid */
    cout << "Error reading Grid: Exiting" << endl;
    exit (-1);
}

/* open the file for dumping summary stats */
filePath = argv[4];
sName = filePath.substr( 0, filePath.find_last_of( DIR_DELIMITER ) + 1 ) + "summary.txt";
sfile.open(sName.c_str());

/* process each input multibeam file in accordance with the input parameters */

for (int j = 0; j < nFiles; j++)
{
  /* create a history for the data file from the input command line. */
  history = "";
  for (int i = 0; i < argc; i++)
  {
```

```
      history += argv[i];
      history += " ";
    }

    /*perform processing  */
    if (type == 1)
    {
      if (processResidual(files[j], g, 4, arg3, history) != 0)
      {
        cout << "Error processing file." << files[j] << endl;
      }
    }
    else if (type == 2)
    {
      if (processRegression(files[j], g, arg3, history) != 0)
      {
        cout << "Error processing file." << files[j] << endl;
      }
    }
    else
    {
      cout << "Error invalid type " << endl << endl;
      usage();
      exit (-1);
    }
  }

  /* Dump a processing summary */
  cout << "Files processed: " << nFiles;
  cout << "Total Samples Read: " << allBeamsRead << endl;
  cout << "Total Samples Considered: " << allBeamsProcessed << endl;
  cout << "Total Samples Invalidated: " << allBeamsFiltered << endl;

  sfile << "Files processed: " << nFiles;
  sfile << "Total Samples Read: " << allBeamsRead << endl;
  sfile << "Total Samples Considered: " << allBeamsProcessed << endl;
  sfile << "Total Samples Invalidated: " << allBeamsFiltered << endl;

  sfile.close();
  free(files);
}
/*******************************************************************************
*
* usage
*
* Purpose: This function dumps the progam calling structure to the screen to
*       inform the user of the applications usage.
*
* Inputs: None
*
* Outputs: None
*
*******************************************************************************/

void usage()
{
  cout << " Usage: MBFilter (-b P|-r Q) gridfile mbfile" << endl << endl;
  cout << " -b performs a simple residual with the input grid invalidating" << endl;
  cout << "    soundings > P percent of water depth away from the grid surface."<< endl;
  cout << " -r performs a regression analysis using a least trimmed squares regression " <<endl;
  cout << "    to eliminate outliers from the multibeam surface followed by a Loess " << endl;
  cout << "    weigthed regression in the local area to determine error at 95% confidence level. " <<endl;
```

```
    cout << "    where Q is the maximum number of pings to use in each regression." <<endl;
    cout << " gridfile name is the name of the gridfile to use as the base reference surface. " << endl;
    cout << " mbfile is the name of the multibeam file in GSF format or a text file with a list " << endl;
    cout << "       of file GSF files.  Note: for a list the first line is expected to be 'LIST'."<< endl << endl;
}

/*****************************************************************************
*
* readFileList
*
* Inputs: fName - the name of the specified file on the input line.
*
* Outputs: files- list of source files to process.
*         nFiles - number of files to process.
*
* Method: 1. Open the file
*         2. Read the first line
*         3. if line = LIST then this is a file list
*         4. else assume it is a GSF file and biuld a list of one file
*
*****************************************************************************/
int readFileList (char *fName, char ***files, int *nFiles)
{
    fstream file;
    char line[1000];
    char **myFiles=NULL;

    /* Open the file */
    file.open(fName,fstream::in );
    if (file.fail())
       return (INPUT_FILE_ERROR);

    /* read the first line and check if it is "LIST"*/
    file.getline(line, 1000);
    if (file.good())
    {
      if (strcmp(line,"LIST") != 0)
      {/* Input is a single GSF file */
        if ((myFiles = (char **) realloc(myFiles, sizeof(char*) * (*nFiles+1))) == NULL)
          return(FILE_MEMORY_ERROR);
        if ((myFiles[*nFiles] = (char *) malloc((size_t) strlen(line))) == NULL)
          return(FILE_MEMORY_ERROR);
        strcpy(myFiles[*nFiles],fName);
        (*nFiles)++;
      }
      else
      { /* This is LIST file with GSF file names in it */
       /* read the file */
        while (!file.eof())
        {
          /*scan the file to get extents, resolution */
          file.getline(line, 1000);
          if (file.good())
          {
           if ((myFiles = (char **) realloc(myFiles, sizeof(char*) * (*nFiles+1))) == NULL)
             return(FILE_MEMORY_ERROR);
           if ((myFiles[*nFiles] = (char *) malloc((size_t) strlen(line))) == NULL)
             return(FILE_MEMORY_ERROR);
           strcpy(myFiles[*nFiles],line);
           (*nFiles)++;
          }
```

```
        }
      }
    }
    else
    {
      return (INPUT_FILE_ERROR);
    }
    file.close();
    *files=myFiles;
    return(0);
}
/*******************************************************************************
*
* processResidual
*
* Inputs: fName - name of input file to process
*       g - grid object to be used as a reference surface
*       order - polynomial order to use for estimating yHat from the gridded surface
*             at the sample x,y
*       percent - threshold to use as validation envelop.  Note this is expressed as a
*             percent of water depth (yHat).
*       history - processing record to log into the source data file
*
* Output: statistics to the text files
*       flags to sample records to mark them invalid
*       history record to source file.
*
* Method:    1. Open GSF file
*          2. While not EOF
*          3.    Get next  MB_PING record
*          4.    If valid ping
*          5.    For each beam
*          6.       if Valid beam
*          7.          Place beam
*          8.          get grid points
*          9.          Estimate Z and beam location
*          10.         if residual > P% estimated Z then
*          11             flag beam as invalid
*          12.         end if
*          13.      end if
*          14.    next beam
*          15. end while
*          16. write history
*          17. close file
*          18. return
*
*******************************************************************************/
int processResidual(char *fName, const grid & g, int order, double percent, string history)
{
    int hnd;
    gsfDataID id;
    gsfRecords gsf_record;
    double lat, lon;
    int i;
    double beamLat, beamLon;
    double along, across;
    double *glats, *glons;
    float **gvals;
    float yHat, dyHat;
    long pingNumber = 0;
    bool nodata = 0;
    bool pingChanged = 0;
```

```
long beamsRead = 0;
long beamsProcessed=0;
long beamsFiltered=0;
string comment;
stringstream ss;
string fStatsName, sfName;
ofstream gFile;

percent = percent / 100.0;

/* malloc the lat, lon, grid value arrays for the polynomial interpolation */
glats = (double *) calloc ((size_t) order, sizeof(double));
glons = (double *) calloc ((size_t) order, sizeof(double));
gvals = (float **) calloc ((size_t) order, sizeof(float *));
for (i = 0; i < order; i++)
{
   gvals[i] = (float *) calloc ((size_t) order, sizeof(float));
}

/* make order even so we have the same number of points north/east and east/west of the point
  for which we are interpolating the value */
if (order%2 > 0) order++;

/* Open GSF file */
if (gsfOpen (fName, GSF_UPDATE_INDEX, &hnd))
{
  gsfPrintError (stderr);
  return (GSF_FOPEN_ERROR);
}

/* open grid vs. sample stats file */
sfName = fName;
fStatsName = sfName + ".stats";
gFile.open(fStatsName.c_str());

/* While not EOF */
/* start at the first ping record */
id.recordID = GSF_RECORD_SWATH_BATHYMETRY_PING;
id.record_number = 1;

/*  Get next  MB_PING record */
while (gsfRead (hnd, GSF_RECORD_SWATH_BATHYMETRY_PING, &id, &gsf_record, NULL, 0) >= 0)
{
   pingNumber++;

   /*  If valid ping */
   lat = gsf_record.mb_ping.latitude;
   lon = gsf_record.mb_ping.longitude;
   if ((lat <= 90.0) && (lon <= 180.0) && !(gsf_record.mb_ping.ping_flags & GSF_IGNORE_PING))
   {
      pingChanged = 0;
      /*  For each beam */
      for (i = 0 ; i < gsf_record.mb_ping.number_beams ; i++)
      {
        /* if Valid beam */
        if (gsf_record.mb_ping.beam_flags != NULL &&
           !(gsf_record.mb_ping.beam_flags[i] & HMPS_IGNORE_NULL_BEAM))
        {
          /* Place beam - get the lats/lons for the sample */
          if ( gsf_record.mb_ping.along_track )
            along = gsf_record.mb_ping.along_track[i];
          else
```

```
    along = 0.0;
  if ( gsf_record.mb_ping.across_track )
    across = gsf_record.mb_ping.across_track[i];
  else
    across = 0.0;

  place_beam(lat, lon, gsf_record.mb_ping.heading, along,
     across, &beamLat, &beamLon);
  beamsRead++;

  /*  get grid points */
  /* Note We are doing a m x m order */
  g.Matrix(beamLat, beamLon, order, glats, glons, gvals);

  /* derease order to 2 if there are no data values in the matrix */
  nodata = 0;
  for (int ii=0;ii<order;ii++)
  {
     for (int jj = 0;jj<order;jj++)
     {
        if (gvals[ii][jj] <= 0.0)
        {
           nodata = 1;
           break;
        }
     }
     if (nodata)
        break;
  }
  if (nodata)
  {
     /*change order to two and test for no data */
     nodata = 0;
     long loc = order/2;
     glats[0] = glats[loc-1]; glats[1]=glats[loc];
     glons[0] = glons[loc-1]; glons[1]=glons[loc];
     if ((gvals [0][0]=gvals[loc-1][loc-1]) <= 0) nodata = 1;
     if ((gvals [1][0]=gvals[loc][loc-1]) <= 0) nodata = 1;
     if ((gvals [0][1]=gvals[loc-1][loc]) <= 0) nodata = 1;
     if ((gvals [1][1]=gvals[loc][loc]) <= 0) nodata = 1;
     order = 2;
  }

  if (!nodata)
  {
    /*  Estimate Z at beam location using polynomial interpolation */
    beamsProcessed++;
    polin2(glons, glats, gvals, order, order, beamLon, beamLat, &yHat, &dyHat);

    /*  if residual > 1% estimated Z then */
    if (fabs(yHat - gsf_record.mb_ping.depth[i]) > yHat * percent)
    {
      /*     flag beam as invalid */
       beamsFiltered++;
       gsf_record.mb_ping.beam_flags[i] |= HMPS_IGNORE_FILTER_EDITED;
       pingChanged = 1;
    } /*   end if */
    else
    {
       if (beamsProcessed % 10)
          gFile << beamsProcessed << "  " << gsf_record.mb_ping.depth[i] << "  " << yHat << endl;
    }
```

```
            } /*end if !nodata */
          } /* end if Valid beam */
        }/* next beam */

        /* if samples are filtered, then save the ping */
        if (pingChanged)
        {
          id.recordID = GSF_RECORD_SWATH_BATHYMETRY_PING;
          id.record_number = pingNumber;
          if (gsfWrite(hnd, &id, &gsf_record) < 0)
          {
            gsfPrintError(stderr);
            exit(-1);
          }
          pingChanged = 0;

        }
      } /*end if valid ping */
      id.recordID = GSF_RECORD_SWATH_BATHYMETRY_PING;
      id.record_number = pingNumber+1;
    } /*end while */

    cout << "Samples Read: " << beamsRead << endl;
    cout << "Samples Considered: " << beamsProcessed << endl;
    cout << "Samples Invalidated: " << beamsFiltered << endl;

    sfile << "Processed: " << fName << endl;
    sfile << "Samples Read: " << beamsRead << endl;
    sfile << "Samples Considered: " << beamsProcessed << endl;
    sfile << "Samples Invalidated: " << beamsFiltered << endl;

    allBeamsRead = allBeamsRead + beamsRead;
    allBeamsProcessed = allBeamsProcessed + beamsProcessed;
    allBeamsFiltered = allBeamsFiltered + beamsFiltered;

    /* Save a history record in the souce data file for pedigree */

    /*  Open the file non-indexed so that we can write a history record.  */
    gsfClose(hnd);
    gFile.close();

    if (gsfOpen (fName, GSF_UPDATE, &hnd))
    {
      gsfPrintError (stderr);
      exit (-1);
    }


    /*  Write a history record describing the filter process.  */

    ss << beamsFiltered;
    comment = "MBFilter: ";
    comment += ss.str();
    comment += " beams invalidated.";

    writeHistory (history, comment, fName, hnd);

    /* close file */
    gsfClose (hnd);

    /* Clean up */
    for (i = 0; i < order; i++)
```

```
    {
        free(gvals[i]);
    }
    free(gvals);
    free(glons);
    free(glats);

    return (0);
}

/********************************************************************************
*
* writeHistory
*
* Inputs: history - processing record to log into the source data file
*         comment - Text comment to be saved in the history record.
*         gsfFile - name of GSF file.
*         handle - file handle
*
* Outputs: history and comment records to source file.
*
* Method:   1. Build the history record
*           2. Seek to the end of the file
*           3. Log the record
*
********************************************************************************/
int writeHistory (string history, string comment, char *gsfFile, int handle)
{
    int         rc;
    int         ret;
    extern int  gsfError;
    time_t      t;
    gsfRecords  rec;
    gsfDataID   gsfID;
    char str[1024];
    char str2[1024];

    memset (&rec, 0, sizeof(rec));

    /* Load the contents of the gsf history record */

    time (&t);
    rec.history.history_time.tv_sec = t;
    rec.history.history_time.tv_nsec = 0;

    strcpy(str, history.c_str());
    rec.history.command_line = str;

    rc = gethostname (rec.history.host_name, sizeof (rec.history.host_name));

    strcpy(str2,comment.c_str());
    rec.history.comment = str2;

    /* Seek to the end of the file and write a new history record */

    ret = gsfSeek (handle, GSF_END_OF_FILE);
    if (ret) cout << "gsfSeek error: " << ret << endl;

    memset (&gsfID, 0, sizeof(gsfID));
    gsfID.recordID = GSF_RECORD_HISTORY;
    ret = gsfWrite (handle, &gsfID, &rec);
```

```
    return(ret);
}
/*****************************************************************************
* processRegression
*
* Inputs: fName - name of input file to process
*       g - grid object to be used as a reference surface
*       pingsToUse - number of consecutive pings to buffer for the Loess regression
*       history - processing record to log into the source data file
*
* Output: statistics to the text files
*       flags to sample records to mark them invalid
*       history record to source file.
*
* Method:   1. Open GSF file
*         2. While not EOF
*         3.   Get next page of pings (pings quantity) of MB_PING records
*         4.   If valid ping
*         5.     For each beam
*         6.       if Valid beam
*         7.         Place beam
*         8          Populate regression arrays with sample
*         9.       end if
*         10.    end for
*         11.  end if
*         12.  perform regression
*         13.  For each beam (sample)
*         14.      if invalidated then set beamflag
*         15.  end for
*         16.  Save GSF Records
*         17. end while
*         18. write history
*         19. close file
*         20. return
*
*****************************************************************************************/
int processRegression(char *fName, const grid & g, double pingsToUse, string history)
{
   int hnd;
   gsfDataID id;
   gsfRecords gsf_record;
   bool endLoop = false;
   int startRec, j, recnum;
   Sample *samples;
   long nsamples;
   double lat, lon;
   int i;
   double beamLat, beamLon;
   double along, across;
   int stat = 0;
   bool flushFlag;

   long pingNumber = 0;
   long beamsRead = 0;
   long beamsProcessed=0;
   long beamsFiltered=0;
   long beamsInAgreement=0;
   double SEPercentofDepthAvg = 0.0;
   string comment;
   stringstream ss;

   double glat, glon;
```

```
float gval;
double dist;
double thresholdScale;

int percent, oldPercent=0;
long pings = (long)pingsToUse;

string iFname;
ofstream iFile;
string fStatsName, sfName;
ofstream gFile;


/* Open GSF file */
if (gsfOpen (fName, GSF_UPDATE_INDEX, &hnd))
{
  gsfPrintError (stderr);
  return (GSF_FOPEN_ERROR);
}

/* open text file for saving difference between TVU and regression standard error if available */
/* open grid vs. sample stats file */
sfName = fName;
fStatsName = sfName + ".gstats";
gFile.open(fStatsName.c_str());
iFname = sfName + ".istats";
iFile.open(iFname.c_str());

/* While not EOF */
/* start at the first ping record */
startRec = recnum = 1;
nsamples = 0;
samples = NULL;

/* process pages of records until end of file */
while (!endLoop)
{
   /* load pings into memory */
   for (j = startRec ; j < startRec + pings ; j++)
   {
     id.recordID = GSF_RECORD_SWATH_BATHYMETRY_PING;
     id.record_number = j;

     if (gsfRead (hnd, GSF_RECORD_SWATH_BATHYMETRY_PING, &id, &gsf_record, NULL, 0) < 0)
     {
       endLoop = true;
       break; /* get out of the for */
     }
     else
     {
      pingNumber++;
     }

     /*  If valid ping */
     lat = gsf_record.mb_ping.latitude;
     lon = gsf_record.mb_ping.longitude;
     if ((lat <= 90.0) && (lon <= 180.0) && !(gsf_record.mb_ping.ping_flags & GSF_IGNORE_PING))
     {

        /* allocate working structure to drive the analysis */
        samples = (Sample *)realloc(samples,(size_t) (nsamples + gsf_record.mb_ping.number_beams) *
                  sizeof(Sample));
```

```
        if (!samples)
        {
          stat = REGRESS_MEMORY_ERROR;
          /* close file */
          gsfClose (hnd);
          return (stat);
        }
        /*  For each beam */
        for (i = 0 ; i < gsf_record.mb_ping.number_beams ; i++)
        {
          /* if Valid beam */
          if (gsf_record.mb_ping.beam_flags != NULL &&
              !(gsf_record.mb_ping.beam_flags[i] & HMPS_IGNORE_NULL_BEAM))
          {
          /* Place beam - get the lats/lons for the sample */
          if ( gsf_record.mb_ping.along_track )
              along = gsf_record.mb_ping.along_track[i];
          else
              along = 0.0;
          if ( gsf_record.mb_ping.across_track )
              across = gsf_record.mb_ping.across_track[i];
          else
              across = 0.0;

          place_beam(lat, lon, gsf_record.mb_ping.heading, along, across, &beamLat, &beamLon);
          beamsRead++;
          /* load regression buffer */
          samples[nsamples].ping = id.record_number;
          samples[nsamples].beam = i;
          samples[nsamples].depth = gsf_record.mb_ping.depth[i];
          samples[nsamples].lat = beamLat;
          samples[nsamples].lon = beamLon;
          samples[nsamples].reg_u = -99999.9;
          if (gsf_record.mb_ping.vertical_error)
              samples[nsamples].tvu = gsf_record.mb_ping.vertical_error[i];
          else
              samples[nsamples].tvu = -99999.9;
          nsamples++;
         } /* end if Valid beam */
       }/* next beam */
     } /*end if valid ping */
  } /* end for  to read in buffer*/
  startRec = j;

  /* do regression */
  if (nsamples > 0)
    stat = regress(&samples, nsamples);

  percent = gsfPercent (hnd);
  if (oldPercent != percent)
  {
    printf ("%3d%% processed    \r", percent);
    fflush (stdout);
    oldPercent = percent;
  }


  /* process results */
  /* Update ping flags for fliers */
  /* Note, sample data are sorted by ping number.  Only write a ping once if at all */
  pingNumber = -1;
  flushFlag = false;
```

```
for (i = 0; i < nsamples; i++)
{
   /* if the sample was identified as a filer then flag the GSF and write it out  */
   if (samples[i].depth <= -99999.0)
   {
      if(samples[i].ping != pingNumber)
      {
         if (pingNumber != -1)
         {
            id.recordID = GSF_RECORD_SWATH_BATHYMETRY_PING;
            id.record_number = pingNumber;

            if (gsfWrite(hnd, &id, &gsf_record) < 0)
            {
               gsfPrintError(stderr);
               exit(-1);
             }
             flushFlag = false;
         }
         id.recordID = GSF_RECORD_SWATH_BATHYMETRY_PING;
         id.record_number = samples[i].ping;

         if (gsfRead (hnd, GSF_RECORD_SWATH_BATHYMETRY_PING, &id, &gsf_record, NULL, 0)
             < 0)
         {
            gsfPrintError (stderr);
            exit (-1);
         }
         pingNumber = samples[i].ping;
      }

      gsf_record.mb_ping.beam_flags[samples[i].beam] |= HMPS_IGNORE_FILTER_EDITED;
      beamsProcessed++;
      beamsFiltered++;
      flushFlag = true;
   }
   else /* get stats on agreement with data and grid */
   {
      g.Point(samples[i].lat, samples[i].lon, &glat, &glon, &gval, &dist);
      if (gval > -99999.0 )
      {
         beamsProcessed++;
         thresholdScale = (1.0 + dist/20.0);
         gFile << i << samples[i].lat << " " << samples[i].lon << " ";
         gFile << samples[i].depth << " " << samples[i].reg_u << "  " << gval;
         gFile << "  " << dist <<" " << (samples[i].reg_u/samples[i].depth) * 100.0;
         gFile << abs(gval - samples[i].depth) <<  endl;
         SEPercentofDepthAvg += ((samples[i].reg_u/samples[i].depth) * 100.0 - SEPercentofDepthAvg)
                                    / (beamsProcessed - beamsFiltered);
         if (abs(gval - samples[i].depth) < thresholdScale * samples[i].reg_u)
         {
            beamsInAgreement++;
         }
      }
      /* output tvu and regression standard error for plotting */
      if (samples[i].tvu >= 0.0)
      {
         iFile << i << " " << samples[i].tvu << " " << samples[i].reg_u << " ";
         iFile << samples[i].depth << " " << samples[i].predDepth << endl;
      }
   }
}
```

```
    /* the last ping may need to be save, if so write it out. */

    if (flushFlag)
    {
      id.recordID = GSF_RECORD_SWATH_BATHYMETRY_PING;
      id.record_number = pingNumber;
      if (gsfWrite(hnd, &id, &gsf_record) < 0)
      {
        gsfPrintError (stderr);
        exit (-1);
      }
      flushFlag = false;
    }

    free(samples);
    samples = NULL;
    nsamples = 0;

  } /*end while !endLoop */

  iFile.close();
  gFile.close();

  cout << "Samples Read: " << beamsRead << endl;
  cout << "Samples Considered: " << beamsProcessed << endl;
  cout << "Samples Invalidated: " << beamsFiltered << endl;
  cout << "Samples in Agreement: " << beamsInAgreement << endl;
  cout << "SE as Percent of Depth: " << SEPercentofDepthAvg << endl;

  sfile << "Processed: " << fName << endl;
  sfile << "Samples Read: " << beamsRead << endl;
  sfile << "Samples Considered: " << beamsProcessed << endl;
  sfile << "Samples Invalidated: " << beamsFiltered << endl;
  sfile << "Samples in Agreement: " << beamsInAgreement << endl;
  sfile << "SE as Percent of Depth: " << SEPercentofDepthAvg << endl;

  allBeamsRead = allBeamsRead + beamsRead;
  allBeamsProcessed = allBeamsProcessed + beamsProcessed;
  allBeamsFiltered = allBeamsFiltered + beamsFiltered;
  allBeamsInAgreement = allBeamsInAgreement + beamsInAgreement;

  /* Save a history record in the source data file for pedigree */

  /* Open the file non-indexed so that we can write a history record. */
  gsfClose(hnd);

  if (gsfOpen (fName, GSF_UPDATE, &hnd))
  {
    gsfPrintError (stderr);
    exit (-1);
  }

  /* Write a history record describing the filter process. */

  ss << beamsFiltered;
  comment = "MBFilter: ";
  comment += ss.str();
  comment += " beams invalidated.";

  writeHistory (history, comment, fName, hnd);
```

```
    /* close file */
    gsfClose (hnd);

    /* Clean up */

    return (0);
}
/* Module regress.cpp */

#include <iostream>
#include <fstream>

#include <gsl/gsl_multifit.h>
#include <gsl/gsl_sort_vector.h>


#include "ellipsoid.h"
#include "geod.h"
#include "MBFilter.h"

static ELLIPSOID ellip=WE; /* WGS84 */

 double getDistance (Sample p1, Sample p2);
 double tricube(double u, double t);
 int leastTrimmedSquare(double data[], long int n, double *mean );
 int ltsQSort(const void *par1, const void *par2);

 /*****************************************************************
 *
 * regress
 *
 * Purpose: This function performs the Least Trimmed Squares regression to
 *          invalidate outliers followed by the Loess weighted regression to
 *          calculate the standard error for each sample.
 *
 * Input: samples - samples to perform the regression techniques on.
 *        n - number of samples.
 *
 * Output: samples - see above.
 *
 * Method: 1. Populate GSL matrices
 *         2. perform Least Squares fit
 *         3. for each sample
 *         4.    set yhat
 *         5  next
 *         6. Perform LTS
 *         7. for each sample
 *         8.    if yhat > scaled mean then invalidate
 *         9. next
 *         10. Reset the GSL arrays without outliers
 *         11. for each sample
 *         12.    calculate the distance to every other sample
 *         13.    sort the distances
 *         14.    for each sample
 *         15.       Calculate sample weight using the tricube function
 *         16.    next
 *         17.    Perform least squares fit
 *         18.    estimate yHat, yErr
 *         19.    Calculate the Standard Error
 *         20. next sample
 *         21. free workspace
 *         22. return
```

```
 *
 * Notes: This function uses the GNU Scientific Library to perform
 *       a weighted least squares fit.
 *
 ****************************************************************/
int regress (Sample **samples, long n)
{
  Sample *samps;
  int i;
  double chisq;
  gsl_matrix *X, *cov;
  gsl_vector *y, *w, *c;
  gsl_vector *xv;
  double y_hat, y_err;
  gsl_vector *dist;
  gsl_vector *sorted_dist;
  double d;
  long n2;
  /*  Loess algorithm parameters. */
  double alpha = 0.33;
  /* Loess neighborhood parameter. */
   const unsigned q = (unsigned) floor(n * alpha);

   /* student t distribution lookup for significant values */
   float t_tab[] = {12.706F, 4.303F, 3.182F, 2.776F, 2.571F, 2.447F, 2.365F, 2.306F, 2.262F, 2.228F,
                    2.201F, 2.179F, 2.160F, 2.145F, 2.131F, 2.120F, 2.110F, 2.101F, 2.093F, 2.086F,
                    2.080F, 2.074F, 2.069F, 2.064F, 2.060F, 2.056F, 2.052F, 2.048F, 2.045F, 2.042F};
   double t, s, s1, se, se1;
   unsigned fliers = 0;
   double XpCX[3];
   long i_index, j_index;
   double sumDepths=0.0;
   double meanDepth;
   double thresholdScale = 2.0;

   X = gsl_matrix_alloc (n, 3);
   y = gsl_vector_alloc (n);
   w = gsl_vector_alloc (n);
   dist = gsl_vector_alloc(n);
   sorted_dist =  gsl_vector_alloc(n);
   xv = gsl_vector_alloc (3);
   c = gsl_vector_alloc (3);
   cov = gsl_matrix_alloc (3, 3);
   gsl_multifit_linear_workspace *work;

   samps = *samples;
   double *win;

   /* load the matrices for the OLS/LTS */

   for (i = 0; i < n; i++)
   {
     gsl_matrix_set (X, i, 0, 1.0);
     gsl_matrix_set (X, i, 1, samps[i].lon);
     gsl_matrix_set (X, i, 2, samps[i].lat);

     gsl_vector_set (y, i, samps[i].depth);
     sumDepths += samps[i].depth;

     /* assign unity to weights for LTS */
     gsl_vector_set (w, i, 1.0);
   }
```

```
work  = gsl_multifit_linear_alloc (n, 3);

/*OLS fit */
gsl_multifit_wlinear (X, w, y, c, cov, &chisq, work);

/* test the fit.  If good then scale the threshold because the mean
   will be small and we don't want to cut out good data */

meanDepth = sumDepths/ n;
if (chisq/n >= meanDepth*.10)
   thresholdScale = 2.0;
else
   thresholdScale = 10.0;

/* remove outliers */
/* LTS */
double mean;
double *yhatdata;

yhatdata = (double *) malloc((size_t) n * sizeof(double));

for (int i = 0; i < n; i++)
{
   gsl_vector_set (xv, 0, 1.0);
   gsl_vector_set (xv, 1, samps[i].lon);
   gsl_vector_set (xv, 2, samps[i].lat);

   gsl_multifit_linear_est (xv, c, cov, &y_hat, &y_err);
   yhatdata[i] = abs(samps[i].depth - y_hat);
}

/*Trim the sum of the resiuals^2*/

leastTrimmedSquare( yhatdata,  n, &mean);

/* Now we have the mean of the best 50% of the residuals */
/* use mean as a filter for outliers */

for (int i = 0; i < n; i++)
{
  gsl_vector_set (xv, 0, 1.0);
  gsl_vector_set (xv, 1, samps[i].lon);
  gsl_vector_set (xv, 2, samps[i].lat);
  gsl_multifit_linear_est (xv, c, cov, &y_hat, &y_err);
  if (abs(samps[i].depth - y_hat) > thresholdScale * mean)
  {
    fliers = fliers + 1;
    samps[i].depth = -99999.9;
  }
}
gsl_multifit_linear_free (work);
free (yhatdata);

/* redo the workspace for n2 if necessary */
n2 = n - fliers;
if (n2 != n)
{
  gsl_matrix_free (X);
  gsl_vector_free (y);
  gsl_vector_free (w);
  X = gsl_matrix_alloc (n2, 3);
```

```
    y = gsl_vector_alloc (n2);
    w = gsl_vector_alloc (n2);

  }
  dist = gsl_vector_alloc(n2);
  sorted_dist =  gsl_vector_alloc(n2);

  /* Do Loess Weighted regression */

  i_index = 0;

  for (i = 0; i < n; i++)
  {
    if (samps[i].depth > -99999.0)
    {
      gsl_matrix_set (X, i_index, 0, 1.0);
      gsl_matrix_set (X, i_index, 1, samps[i].lon);
      gsl_matrix_set (X, i_index, 2, samps[i].lat);

      gsl_vector_set (y, i_index, samps[i].depth);
      i_index++;
    }
  }

  work  = gsl_multifit_linear_alloc (n2, 3);
  win = (double *) malloc((size_t) n2 * sizeof(double));

  i_index = 0;
  for (int i = 0; i < n; i++)
  {
    if (samps[i].depth > -99999.0)
    {
      j_index = 0;
      /* Compute distances. */
      for (int j = 0; j < n; j++)
      {
        if (samps[j].depth > -99999.0)
        {
          d = getDistance(samps[i], samps[j]);
          gsl_vector_set(dist, j_index, d );
          gsl_vector_set(sorted_dist, j_index, gsl_vector_get(dist, j_index));
          j_index++;
        }
      }

      /* Sort distances in order from smallest to largest. */
      gsl_sort_vector(sorted_dist);

      /* Compute weights using the tricube weight function. */
      for (int j = 0; j < n2; j++)
      {
        win[j] = tricube(gsl_vector_get(dist, j), gsl_vector_get(sorted_dist, q));
        gsl_vector_set (w, j, win[j]);
      }

      gsl_multifit_wlinear (X, w, y, c, cov, &chisq, work);
      gsl_vector_set (xv, 0, 1.0);
      gsl_vector_set (xv, 1, samps[i].lon);
      gsl_vector_set (xv, 2, samps[i].lat);

      gsl_multifit_linear_est (xv, c, cov, &y_hat, &y_err);
      samps[i].predDepth = y_hat;
```

```
      /* compute the compute confidence limits on mean y at lat, lon */
      /* yHat +- t(q-2-1, 0.975) * s * sqrt(Xprime * C * x), see Draper, smith pg, 210 */
      if (q < 4) /*too few samples */
        t = -1.0;
      else if (q-4 <= 30)
        t = t_tab[q-4];
      else if (q-4 <= 60)
        t = 2.0;
      else
        t = 1.96;
      if (t > 0.0)
      {
        s = sqrt(chisq / (q-2));
        for (unsigned j = 0; j < 3; ++j)
        {
          XpCX[j] = 0.0;
          for (unsigned k = 0; k < 3; k++)
            XpCX[j] += gsl_vector_get (xv, k) * gsl_matrix_get(cov,j,k);
        }
        s1 = 0.0;
        for (unsigned j = 0; j < 3; ++j)
        {
          s1 += XpCX[j] * gsl_vector_get (xv, j);
        }
//                      se = t * s * sqrt(s1);
        se1 = t * s * sqrt (1 + s1);
        samps[i].reg_u = se1;
      } /* t > 0.0 */
    } /* if samps[i] > -99999 */
  } /* for  i < n */

  free(win);
  gsl_multifit_linear_free (work);

  gsl_vector_free(sorted_dist);
  gsl_vector_free(dist);

  gsl_vector_free(xv);
  gsl_matrix_free (X);
  gsl_vector_free (y);
  gsl_vector_free (w);
  gsl_vector_free (c);
  gsl_matrix_free (cov);

  return 0;
}

/*************************************************************
*
* getDistance
*
* Purpose: This function calculates the distance between points.
*
* Input: p1, p2 - sample points.
*
* Output: none.
*
* Method:
*
* Notes: This function uses the Bathy Hydro Post Processing Library
*       function geo_inverse.
```

```
 *
 ****************************************************************/

 double getDistance (Sample p1, Sample p2)
 {
    double az, az2, dist;

    geo_inverse( AXIS(ellip), RFLAT(ellip), p1.lat,p1.lon,p2.lat,p2.lon, &az, &az2, &dist);

    return (dist);

 }

 /*************************************************************
 *
 * tricube
 *
 * Purpose: This function calculates a weight (0.0 - 1.0) depending
 *          on distance from the point for the Loess by applying the triciube
 *          weight function.
 *
 * Input: t  - distance entry in sorted distance array.
 *        u - distance entry in unsorted distance array.
 *
 * Output: none.
 *
 * Method:
 *
 ****************************************************************/

double tricube(double u, double t)
{
   // 0 <= u < t

   if ( 0.0 <= u && u < t)
   {
      // (1 - (u/t)^3)^3
      return pow( ( 1.0 - pow(u/t, 3)), 3 );
   }
   // u >= t
   else {
      return 0.0;
   }

}

 /*************************************************************
 *
 * leastTrimmedSaure
 *
 * Purpose: This sorts the residuals from a least squares regression,
 *          then returns the mean of the smallest half samples.
 *
 * Input: data - sample array of residuals.
 *        n - number of samples.
 *
 * Output: mean- the mean of the smallest 1/2 of the residuals.
 *
 * Method: 1. sort the data
 *         2. Loop through the first half accumulating the sum a the sum of the squares
 *         3. ensure we have the minimum comparing 1/2 the values at a time.
 *
```

```
 * Notes: The algorithm (exactly) is described in P.J. Rousseeuw and A.M. Leroy:
 *       Robust Regression and Outlier Detection John Wiley & Sons 1987. This
 *       implementation was performed by N. Devillard with Public Domain rights.
 *
 ***********************************************************************/

int leastTrimmedSquare( double data[], long int n, double *mean )
{
  int i,j,h,h2;
  double score,best_score,loc,best_loc,old_sum,new_sum;
  double old_power_sum,new_power_sum;

  h = n - n/2;
  h2 = n/2;

  qsort(data, n, sizeof(double),ltsQSort);


  old_sum = 0;
  old_power_sum = 0.0;
  for(i = 0;i < h;i++)
  {
    old_sum = old_sum + data[i];
    old_power_sum = old_power_sum + data[i]*data[i];
  }

  loc = old_sum/h;
  /* For better understanding of the algorithm:
  O(N^2) implementation of the algorithm would compute score as:
  score = 0.0;
  for(i = 0;i < h;i++)
  {
    score = score + (data[i] - loc)*(data[i] - loc);
  }
  But there is a faster way to this: */

  score = old_power_sum - old_sum*loc;

  best_score = score;
  best_loc = loc;

  for(j = 1;j < h2 + 1;j++)
  {
    new_sum = old_sum - data[j - 1] + data[h - 1 + j];
    old_sum = new_sum;
    loc = old_sum/h;
    new_power_sum = old_power_sum - data[j - 1]*data[j - 1] + data[h - 1 + j]*data[h - 1 + j];
    old_power_sum = new_power_sum;
    score = old_power_sum - old_sum*loc;

    if(score < best_score)
    {
      best_score = score;
      best_loc = loc;
       i = j-1;
    }
  }
  *mean = best_loc;


 return(0);
}
```

```
int ltsQSort(const void *par1, const void *par2)
{
  if( *((double*)par1) < *((double*)par2)) return(-1);
  else if( *((double*)par1) > *((double*)par2)) return(1);
  else return(0);
}
/* module grid.cpp */

#include "ellipsoid.h"
#include "geod.h"
#include "grid.h"

ELLIPSOID ellip = WE; /*we will use WGS-84 for positioning */

/*********************************************************************
*
* grid::grid
*
* Purpose: Constructor for a new grid object
*
* Inputs: None
*
* Outputs: none
*
* Method: 1. initialize public and private properties

*********************************************************************/
grid::grid(void)
{
  northLat = southLat = westLon =  eastLon= 0.0;
  xRes =  yRes = 0.0;
  nx =  ny = 0;
  resUOM = 0;  /* default to geographic */

  values = NULL;
  nValues = 0;
  fileType = 0;

}

/*********************************************************************
*
* grid::~grid
*
* Purpose: Destructor for a grid object
*
* Inputs: None
*
* Outputs: none
*
* Method: 1. free memory allocated for the grid data

*********************************************************************/

grid::~grid(void)
{
  if (values)
  {
    for (long i=0; i < nx; i++)
    {
```

```
            free(values[i]);
        }
        free(values);
    }
}

/*****************************************************************************
*
* ingestGrid
*
* Purpose: This function reads a grid file from disk.  It populates the grid post
*        values and sets properties used for hashing into the grid to appropriately
*        retrieve values.
*
* Input: fName - name of the grid file.
*
* Output: none
*
* Method: 1. Open the file
*           2. if Failure, then return GRID_ERROR_CANNOT_OPEN_SOURCE
*           3. The format of file to read depends on the file extension.  The types that are read are:
*                   1. ASCII XYZ, no header
*           4. if ASCII XYZ then read file populating values and properties as we go
*                   The following assumptions apply:
*                   1. Data are in  X - Longitude, Y - Latitude, and Z - Depth value
*                   2. Data are row major order, thus the longitude varies first followed by latitude
*                   3. Latitude and Longitude are in decimal degrees
*                   4. Resolution must be determined X-X and Y-Y.  If these are not integer values
*                      then the resolution is determined to be in meters and the Grid is in some
*                      projected space.
*
*****************************************************************************/
int grid::ingestGrid(char *fName)
{
   int len = 0;
   int errorCode=0;

   /* Determine the file type using the extension */
   len = strlen(fName);
   if (strcmp(fName + (len - 3), "XYZ") == 0 ||
      strcmp(fName + (len - 3), "xyz") == 0)
      fileType = 1;

   /* Open the file */
   file.open(fName);
   if (file.fail())
      return (GRID_ERROR_CANNOT_OPEN_SOURCE);

   /* read the file */
   if (fileType == 1)
   { /*read XYZ */
      char line[1000];
      char *sval;
      int count = 0;
      double lon=-1111.0, lat=-1111.0;
      float val = 0.0;
      double prevLon, prevLat;
      bool firstRow = 1;
      long recs=0;

      southLat = westLon = 9999.9;
      northLat = eastLon = -9999.9;
```

```
    while (!file.eof())
    {
      /*scan the file to get extents, resolution */
      file.getline(line, 1000);
      if (file.good())
      {
        prevLon = lon;
        prevLat = lat;
        sval = line;
        lon = strtod(sval,&sval);
        lat = strtod(sval,&sval);
        val = (float) strtod(sval,&sval);
        recs++;
        if (lon < westLon)
          westLon = lon;
        if (lon >eastLon)
          eastLon = lon;
        if (lat < southLat)
          southLat = lat;
        if (lat > northLat)
          northLat = lat;
        if (recs == 2)
          xRes = (float) abs(lon-prevLon);
        if (recs > 2 && prevLat != lat)
          yRes = (float) abs(lat - prevLat);
      }
    }
    cout << "Records read: "<< recs << endl;
    cout << "extents:        " << northLat << endl;
    cout <<"        "<< westLon << "      " << eastLon << endl;
    cout << "                " << southLat << endl;
    cout << " XRes = " << xRes << " yRes = " << yRes << endl;
    cout << endl << "Ingesting data points" << endl;
    /* this needs to be precise and strtod is letting me down */
    ny = (long) floor(((northLat - southLat) / yRes) + 0.5) + 1;
    nx = (long) floor(((eastLon - westLon) / xRes) + 0.5) + 1;
    cout << "rows = " << ny  << endl;
    cout << "columns = " << nx << endl;

    file.close();
    file.open(fName);
    if (file.fail())
      return (GRID_ERROR_CANNOT_OPEN_SOURCE);

    if ((errorCode = allocateZ()) != 0)
      return(errorCode);
    while (!file.eof())
    {
      file.getline(line, 1000);
      if (file.good())
      {
        sval = line;
        lon = strtod(sval,&sval);
        lat = strtod(sval,&sval);
        val = (float) strtod(sval,&sval);
        if ((errorCode = addZ(lat, lon, val)) !=0)
          break;
      }
    }
}

cout << " Grid ingested ." << endl;
```

```
    file.close();
    return(errorCode);
}
/*****************************************************************************
*
* grid::addZ
*
* Purpose: This method adds data values to the grid.
*
* Input: y - latitude coordinate.
*        x - longitude coordinate.
*        Z - Depth value.
*
* Output: none
*
* Method: 1. calculate row, column indices
*         2. For me depths are positive, so swap sign if necessary.
*                    3. Set the grid value.
*
*****************************************************************************/
int grid::addZ(double y, double x, float Z)
{
    long row, col;

    row = (long) floor(((y - southLat) / yRes) + 0.5);
    col = (long) floor(((x - westLon) / xRes) + 0.5);

    if (Z < 0.0f) Z = Z * -1.0f;
    values[col][row] = Z;

    return(0);
}

/*****************************************************************************
*
* grid::allocateZ
*
* Purpose: This method allocates the needed memory for the grid after nx and ny are set.
*
* Input: none
*
* Output: none
*
* Method: 1. allocate columns
*           2. for each column, allocate the rows
*
*****************************************************************************/
int grid::allocateZ()
{
    if ((values = (float **)calloc(nx,sizeof(float*))) == NULL)
        return(GRID_MEMORY_ERROR);
    for (long i=0; i < nx; i++)
    {
        if ((values[i] = (float *)calloc(ny,sizeof(float))) == NULL)
          return(GRID_MEMORY_ERROR);
        memset(values[i], 0, (size_t)ny);
    }

    return(0);
}
```

```
/*****************************************************************************
*
* grid::Matrix
*
* Purpose: This method returns a set of grid values surrounding a point.
*
* Input: lat - the y coordinate of the point of interest.
*        lon - the x coordinate of the point of interest.
*        order - the size of the areas to return (rows and columns).
*
* Output: lats - vector of the latitudes being returned.
*         lons - vector of the longitudes being returned.
*         vals - matrix of the depths be returned.
*
* Method: 1. Determine where in grid the point is
*         2. Set the start x and y
*         3. Load the sub grid reducing order if we are on the edge of the grid
*
*****************************************************************************/
int grid::Matrix(double lat, double lon, int order, double *lats, double *lons, float **vals) const
{

  long row, col;
  long ix, iy;
  long startx, starty;

  row = (long) floor(((lat - southLat) / yRes) + 0.5);
  col = (long) floor(((lon - westLon) / xRes) + 0.5);

  startx = (col - order / 2);
  starty = (row - order / 2);

  for (ix = 0; ix < order; ix++)
  {
    lons[ix] = westLon + ((startx + ix)  * xRes);
    for (iy = 0; iy < order; iy++)
    {
      if ((startx + ix) < 0 || (startx + ix) > nx -1 ||
          (starty + iy) < 0 || (starty + iy) > ny -1)
        vals[ix][iy] = -1.0;
      else
        vals[ix][iy] = values[startx + ix][starty + iy];
      if (ix == 0)
        lats[iy] = southLat + ((starty + iy) * yRes);
    }

  }
  return (0);
}

/*****************************************************************************
*
* grid::Point
*
* Purpose: This method returns closest grid node to a point.
*
* Input: lat - the y coordinate of the point of interest.
*        lon - the x coordinate of the point of interest.
*
* Output: glat - the latitude of the closest grid point.
*         glon - the longitude of the closest grid point.
```

```
*          val - the depth at the closest grid node.
*         dist - distance from the point of interest to the grid node.
*
* Method: 1. calculate the grid cell
*         2. calculate distance to each of the four surrounding grid posts and
*            return the minimum.
*
*****************************************************************************/
int grid::Point(double lat, double lon, double *glat, double *glon, float *val, double *dist) const
 {

   long row, col;
   double d;
   double lat1, lon1;
   double az1, az2;

   *val = -99999.9F;
   *dist = 99999.9;

   if (lat < southLat || lat > northLat)
      return ( GRID_OUT_OF_BOUNDS);
   if (lon < westLon || lon > eastLon)
      return ( GRID_OUT_OF_BOUNDS);

   row = (long) floor(((lat - southLat) / yRes) + 0.5);
   col = (long) floor(((lon - westLon) / xRes) + 0.5);

   /* look at the fouyr surrounding points and return the closest one */
   lat1 = (southLat + (row * yRes));
   lon1 = (westLon + (col * xRes));
   geo_inverse(AXIS(ellip), RFLAT(ellip), lat, lon, lat1, lon1,   &az1, &az2, &d );
   if (d < *dist )
   {
      *dist = d;
      *glat = lat1;
      *glon = lon1;
      *val = values[col][row];

   }
   lon1 = lon1 + xRes;
   geo_inverse(AXIS(ellip), RFLAT(ellip), lat, lon, lat1, lon1,   &az1, &az2, &d );
   if (d < *dist )
   {
      *dist = d;
      *glat = lat1;
      *glon = lon1;
      *val = values[col+1][row];
   }
   lat1 = lat1 +  yRes;
   geo_inverse(AXIS(ellip), RFLAT(ellip), lat, lon, lat1, lon1,   &az1, &az2, &d );
   if (d < *dist )
   {
      *dist = d;
      *glat = lat1;
      *glon = lon1;
      *val = values[col+1][row+1];
   }
   lon1 = lon1 - xRes;
   geo_inverse(AXIS(ellip), RFLAT(ellip), lat, lon, lat1, lon1,   &az1, &az2, &d );
   if (d < *dist )
   {
      *dist = d;
```

```
        *glat = lat1;
        *glon = lon1;
        *val = values[col][row+1];
    }
    return (0);
}

/* MBFilter.h */

#ifndef __MBFILTER_H__
#define __MBFILTER_H__
#include <sstream>
#include <string>

#ifdef WIN32
#include <WinSock.h>
#else
#include <unistd.h>
#endif
#include "hmpsflag.h"
#include "grid.h"
#include "polin.h"

#ifdef WIN32
#define DIR_DELIMITER '\\'
#else
#define DIR_DELIMITER '/'
#endif


#define INPUT_FILE_ERROR -201
#define FILE_MEMORY_ERROR -202
#define REGRESS_MEMORY_ERROR -203

typedef struct {
    double lat;
    double lon;
    long ping;
    long beam;
    double depth;
    double predDepth;
    double tvu;
    double reg_u;
} Sample;

void usage();
int place_beam( double ping_lat, double ping_lon, double ping_hdg,
        double along_track, double across_track,
        double *beam_lat, double *beam_lon );
int processResidual(char *fName, const grid &g, int order, double percent, string history);
int processRegression(char *fName, const grid &g, double pings, string history);
int regress(Sample **samples, long nsamples);
int writeHistory (string history, string comment, char *gsfFile, int handle);
int readFileList (char *fName, char ***files, int *nFiles);


#endif
```

```
/* Grid .h */

#ifndef __GRID_H__
#define __GRID_H__
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>

/* Error codes associated with gridded data */
using namespace std;

#define GRID_ERROR_CANNOT_OPEN_SOURCE -101
#define GRID_MEMORY_ERROR -102
#define GRID_OUT_OF_BOUNDS -103


class grid
{
private:
        double northLat, southLat,westLon, eastLon; /*Extents of the grid */
        float xRes, yRes;                    /* resolution in x (Lon) direction */
                                             /*resolution in y (Lat) direction */
        long nx, ny;                         /* number of cells/post in x and y */
        int resUOM;                          /* Unit of measure on the X/Y Axis */
                                             /* 0 - geographic in decimal minutes */
                                             /* 1 - geographic in decimal degrees */
                                             /* 2 - projected in meters */
        float **values;                      /* the grid values */
                                             /* Row major order addressed from the
                                                south-western corner */
        long nValues;                        /* number of entries in grid.  used for memory management */
        fstream file;                        /* input stream for file */
        int fileType;                        /* Format indicator for the Grid file type
                                                0 - Unknown
                                                1 - XYZ ASCII */
        int addZ(double lat, double lon, float Z);
        int allocateZ();
public:
        grid(void);
        ~grid(void);
        int ingestGrid(char *fName);
        int Matrix(double lat, double lon, int order, double *lats, double *lons, float **vals) const;
        int Point(double lat, double lon,  double *glat, double *glon, float *val, double *dist) const;
};

#endif
```

REFERENCES

Becker JJ., Sandwell DT., Smith WH., Braud J., Binder B., Depner J., Fabre D., Factor J., Ingalls S., Kim S., Ladner RW., and others. 2009. Global Bathymetry and Elevation Data at 30 Arc Seconds Resolution: SRTM30_PLUS. Marine Geodesy 32(4):355-371. DOI:10.1080/01490410903297766.

Calder BR., Byrne JS., Lamey W., Breenan R., Case JD., Fabre D., Ladner RW., Moggert F., Gallaghe B. 2005. The Open Navigation Surface Project. International Hydrographic Review 2005.

Calder BR. 2006. On the Uncertainty of Archive Hydrographic Datasets. IEEE Journal of Oceanic Engineering 31(2): 249-265. DOI: 10.1109/JOE.2006.872215

Calder BR., Mayer LA. 2003. Automatic processing of high-rate, high-density multibeam echosounder data. Geochemistry. Geophysics. Geosystems., 4(6):1048. doi:10.1029/2002GC000486.

Chandler MT., Wessel P. 2008. Improving the quality of marine geophysical track line data: Along-track analysis, J. Geophys. Res. 113(B02102). DOI: 10.1029/2007JB005051.

Cleveland W., Devlin S. 1988. Locally Weighted Regression: An Approach to Regression Analysis by Locally Fitting. J of Am Statistical Assoc 83(403): 596–610.

Cormen T., Leisierson C., Rivest R., Stein C. 2009. Introduction to Algorithms. Third Edition. Cambridge (USA): The MIT Press. 1292 p.

Draper N., Smith H. 1981. Applied Regression Analysis, Second Edition. New York: John Wiley and Sons. 709 p.

Elmore P., Fabre D. Sawyer R., Ladner R. 2012. Uncertainty Estimation of Historical Bathymetric Data from Bayesian Networks. Geochemistry Geophysics Geosystems 13(Q09011):11. doi:10.1029/2012GC004144.

Farr, H. 1980. Multibeam Bathymetric Sonar: Sea Beam and Hydro Chart. Marine Geodesy 4(2):77-93. doi:10.1080/15210608009379375.

Hare R., Godin A., Mayer L.A. 1995. Accuracy estimation of Canadian swath (Multi-beam) and sweep (Multi-Transducer) sounding systems, Tech. rep., Canadian Hydrographic Service, Ottawa, Ont.

Jakobsson M., Mayer LA., Coakley B., Dowdeswell JA., Forbes S.,Fridman B., Hodnesdal H., Noormets R., Pedersen R., Rebesco M., and others. The International Bathymetric Chart of the Arctic Ocean (IBCAO) Version 3.0. Geophysical Research Letters. *doi: 10.1029/2012GL052219*

Press W., Teukolsky S., Vetterling W., Flannery B. 1992. Numerical Recipes in C. 2nd ed. New York: Cambridge University Press. p 108-125.

Rousseeuw P., Leroy A. 1987. Robust Regression and Outlier Detection. New York: John Wiley & Sons, Inc. 313 p.

Ryan WB., Carbotte SM., Coplan JO., O'Hara S., Melkonian A., Arko R., Weissel RA., Ferrini V., Goodwillie A., Nitsche F., and others. 2009. Global Multi-Resolution Topography synthesis. Geochem. Geophys. Geosyst. 10(3). doi:10.1029/2008GC002332.

Smith WH., Sandwell DT. 1997 Global sea floor topography from satellite altimetry and ship depth soundings. Science 277:1956-1962.

Taylor, Barry, and C. Kuyatt (1994), NIST Technical Note 1297; Guidelines for Evaluating and Expressing the Uncertainty of NIST Measurement Results. Physics Laboratory; National Institute for Standards and Technology

Wessel P., Chandler M. 2011. The Spatial and Temporal Distribution of Marine Geophysical Surveys. Acta Geophysica 59(1). DOI:10.2478/s11600-010-0038-1

West M., Harrison J. 1997. Bayesian Forecasting and Dynamic Models 2 ed., New York: Springer-Verlag 700 p.

[Free Software Foundation]. 2011 Jul 03. GNU Scientific Library – Reference. <http://www.gnu.org/software/gsl/manual>. 2012 November 17.

[IHO]. 2008 Feb. IHO Standards for Hydrographic Surveys; Special Publication No. 44. Fifth Edition. Monaco: International Hydrographic Bureau. 28 p.

[JCGM]. 2008. 100:2008 Evaluation of measurement data – Guide to the expression of uncertainty in measurement. < http://www.bipm.org/utils/common/documents/jcgm/JCGM_100_2008_E.pdf>. 134 p.

[NIST]. 2012 April. NIST/SEMATECH e-Handbook of Statistical Methods, <http://www.itl.nist.gov/div898/handbook/>. 2012 November 17.